DOCTOR OF SCIENCES

Algorithms and software for multilevel nonlinear optimization

Tomanos, Dimitri

Award date:
2009

Awarding institution:
University of Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX NAMUR

FACULTE DES SCIENCES

DEPARTEMENT DE MATHEMATIQUE

Algorithms and Software for Multilevel Nonlinear Optimization

Dissertation présentée par
Dimitri Tomanos
pour l'obtention du grade
de Docteur en Sciences

Composition du Jury:

Serge GRATTON
Dominique ORBAN
Annick SARTENAER
Philippe TOINT (Promoteur)
Joseph WINKIN

2009

Imprimé en Belgique

ISBN : 978-2-87037-646-1
Dépôt légal: D / 2009 / 1881 / 34

Facultés Universitaires Notre-Dame de la Paix
Faculté des Sciences
rue de Bruxelles, 61, B-5000 Namur, Belgium

Facultés Universitaires Notre-Dame de la Paix
Faculté des Sciences
Rue de Bruxelles, 61, B-5000 Namur, Belgium

Algorithmique et logiciel dans les problèmes d'optimisation non-linéaire multi-niveaux

par Dimitri Tomanos

Résumé: Cette recherche concerne l'étude algorithmique des problèmes d'optimisation non-linéaire multi-niveaux. Nous avons développé au cours de cette thèse de nouvelles méthodes pour la résolution de ce type de problèmes et nous analysons leur convergence et résultats numériques. Nous avons également implémenté un logiciel pour lequel une description complète est donnée dans ce manuscrit. Une librairie de problèmes tests a également été construite sur laquelle nous avons pu tester notre logiciel. Les résultats obtenus montrent clairement que notre méthode est la meilleure pour cette classe de problèmes. Le logiciel a également permis la résolution d'une application industrielle de verres progressifs qui n'avaient pu être résolue par les méthodes existantes.

Algorithms and software for multilevel nonlinear optimization

by Dimitri Tomanos

Abstract: This research concerns the algorithmic study of multilevel nonlinear optimization problems. We have developped during our thesis new methods to solve this type of problems and we analyze their convergence and numerical results. We have also implemented a software for which a complete description is given in this manuscript. A library of test problems has also been constructed on which we have tested our software. The numerical results obtained show that our method is clearly the best for this class of problems. The software also allowed the solution of a progressive lens industrial application that had never been solved by existing methods.

Dissertation doctorale en Sciences mathématiques (Ph.D. thesis in Mathematics)
Date: 7-05-2009
Département de Mathématique
Promoteur (Advisor): Prof. Ph. L. TOINT

Remerciements

Tout d'abord, je tiens à remercier mon promoteur, Philippe Toint, pour son encadrement ainsi que pour son engouement pour la recherche qu'il a pu me transmettre lors de ces années de collaboration. Son soutien, son expérience ainsi que ses remarques pertinentes ont permis l'aboutissement de ce projet de recherche. Je le remercie également pour m'avoir toujours permis d'orienter ma recherche selon mes affinités ainsi que pour m'avoir donné l'opportunité de participer à plusieurs conférences internationales et de m'y avoir fait rencontrer plusieurs chercheurs et professeurs avec lesquels j'ai pu avoir des discussions très intéressantes.

Ma reconnaissance s'adresse également au Fonds pour la Formation à la Recherche dans l'Industrie et dans l'Agriculture (FRIA), qui m'a accordé une bourse de doctorat et m'a ainsi permis de mener à bien mon travail de thèse.

Je tiens également à remercier Annick Sartenaer et Serge Gratton pour leur collaboration à une partie de ce travail. Je les remercie également, ainsi que Joseph Winkin et Dominique Orban, d'avoir accepté de faire partie du jury de cette thèse. Je tiens également à tous les remercier pour leurs remarques et suggestions pertinentes qui ont permis d'améliorer ce manuscrit.

Merci à Melissa Weber Mendonça, collègue de bureau et de travail, pour son amitié, sa bonne humeur, les cours de portugais, les nombreux cafés et les conversations de geek pendant ces années dans le même bureau.

Mes plus chaleureux remerciements s'adressent également à tous mes collègues sans qui ces quatre années auraient été sans nul doute fort différentes. Je tiens à remercier mes collègues de l'unité d'analyse numérique avec qui j'ai partagé plusieurs discussions de travail, plusieurs conférences mémorables ainsi que plusieurs fou rires. Merci à Katia Demaseure, Selime Gurol, Patrick Laloyaux, Laetitia Legrain, Vincent Malmedy, Caroline Sainvitu, Jean Tshimanga et Emilie Wanufelle. Je tiens à remercier les habitués du café du matin Eric Cornelis, Nicolas Delsate et Benoît Noyelles. Je remercie également les amis du whist, Joffray Baune, Charlotte Beauthier, Jehan Boreux, Fabrice Calay, Audrey Compere, Sandrine d'Hoedt, Anne-Sophie Libert et Sebastian Xhonneux pour toutes ces parties mémorables partagées. Je tiens enfin à remercier tous les autres scientifiques que j'ai cotoyés pendant mes années au département de mathématique pour ces moments partagés. Merci à Johan Barthelemy, Marie Castaigne, Martine De Vleeschouwer, Julien Descamps, Julien Dufey, Nicolas Franco, André Fuzfa, Xavier Pauly, Vincent Piefort, Simone Righi, Stéphane Valk et Fabien Walle. Mes remerciements vont aussi vers tous les autres membres du département de Mathématique pour leur accueil et leur convivialité durant ces années.

Merci aussi à tous mes amis, mathématiciens ou non, qui m'ont aidé, par tous ces moments de détente passés en leur compagnie.

Je tiens à remercier Mélodie Mouffe, amie de toujours, pour les moments partagés depuis la première candi. Merci pour ses conversations téléphoniques interminables, ces mails kilométriques et pour cette amitié.

Je tiens également à remercier toute ma famille pour son soutien pendant ces années. Je tiens tout particulièrement à remercier mon père qui m'a toujours montré son soutien et sa fierté pour mon travail. J'ai également une pensée tout particulière pour ma mère, trop tôt disparue, sans qui je ne serais ce que je suis.

Enfin, mon dernier remerciement s'adresse à ma femme, Sabrina. Merci d'avoir toujours été présente, d'avoir toujours patiemment essayé de me comprendre, de m'avoir toujours écouté et d'avoir toujours fait tout ce qui était possible pour me rendre heureux. Sans son amour, je n'aurais jamais pu mener ce projet à bien.

A vous tous, merci pour tout ce que vous m'avez apporté,

Dimitri

Contents

Introduction

Optimization is the branch of mathematics which studies the means to obtain the best possible value of some function. This topic has become a very important area of research for more than fifty years motivated especially by the ever-growing needs of industry.

The optimization of finite-dimensional discretizations of problems in infinite dimensional spaces has been considered widely in the last years. New interest in surface design, data assimilation for weather forecasting (see Fisher (1998)) or in optimal control of systems described by partial-differential equations have been the main motivation of this challenging research trend, but other applications such as multi-dimensional scaling (Bronstein, Bronstein, Kimmel and Yavneh, 2005) or quantization schemes (Emilianenko, 2005) also give rise to similar questions. While the direct solution of such problems for a discretization level yielding the desired accuracy is often possible using existing packages for large-scale numerical optimization, this technique typically does make very little use of the fact that there is an underlying infinite-dimensional problem for which several discretization levels are possible, and the approach thus rapidly becomes cumbersome. This observation motivates the developments of multilevel optimization that makes explicit use of this fact in the hope to allow better efficiency and, possibly, enhance reliability.

This thesis is concerned with the algorithmic study of multilevel optimization, in particular in the context of trust-region techniques.

Structure of the document

We divided this document into four chapters. In the first Chapter, after the introduction of basic concepts about optimization, we introduce the trust-region method and some refinements we use in the following chapters.

The major concepts of multilevel optimization are borrowed from the multigrid methods for linear systems. We thus start our second Chapter with the exposition of these methods. We then present how these concepts have been applied to optimization in linesearch methods and in the adaptive cubic overestimation technique. We then present in details how we have adapted the multilevel philosophy to trust-region methods by presenting two different techniques.

One of the multilevel trust-region techniques presented in Chapter 2, the RMTR (Recursive Multilevel Trust Region) algorithm, performs clearly better than the other one. We have thus implemented a complete software package using this method. In Chapter 3, we present the algorithmic details of the method and how the software can be used.

In Chapter 4, we present a library of multilevel problems we have constructed to test our multilevel software. We then present the numerical results obtained with our software on this library. The software also permits the solution of an industrial problem of progressive adaptive lenses. This application and the numerical results are presented at the end of the chapter.

We finally conclude and give some perspectives of future work. A summary of our contributions and tables containing the main notations and abbreviations used are also presented after the conclusion.

Chapter 1

Nonlinear optimization

1.1 What is optimization?

Optimization is the branch of mathematics which studies the means to obtain the best possible value of some function. This topic is quite large and appears every day in human life. For example, a factory tends to maximize its profit and/or minimize its costs. Even nature optimizes, a physical closed system always tends to its state of minimum internal energy (principle of minimum energy).

To correctly define an *optimization problem*, or sometimes called a *mathematical program*, we need an *objective function* which is a measure of the quality of the system under study and which depends on a set of *variables* whose values may be chosen. We may also need a set of *constraints* which are conditions to impose to the variables to have a problem well-defined. These constraints correspond to restrictions on the variables or on their interrelations. The purpose of an optimization problem is then to find values for the variables which satisfy the constraints and for which the quality of the system is as large as possible (which for historical reasons has to be achieved by making the objective function as small as possible).

The construction of these ingredients from a real problem is called *modelization*. This is a very important phase in an optimization process. The real problem is often very complex and this phase selects the most important characteristics of it to create a *model* which is then optimized with an optimization algorithm. The solution of this model is then brought back to the real world problem. Since the model is not an exact representation of the real problem, the solution may not correspond to the best solution of the real problem. The model may thus be improved up to a state where it is validated and the solution is then interpreted as a solution of the real problem.

Mathematically, an *unconstrained optimization problem* is formulated as

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function and $x \in \mathbb{R}^n$ are the variables. For unconstrained optimization problems, no constraints are imposed on the variables. In *constrained optimization problems*, the variables x are required to belong to a *feasible region* Ω defined as

$$\Omega = \{x \in \mathbb{R}^n \mid c_i(x) \leq 0,\ i \in \mathcal{I} \quad \text{and} \quad c_i(x) = 0,\ i \in \mathcal{E}\} \tag{1.2}$$

where $\mathcal{E}$ and $\mathcal{I}$ are index sets and $c_i : \mathbb{R}^n \to \mathbb{R}$ $(i \in \mathcal{E} \cup \mathcal{I})$ are the constraints of the problem. The sets $c_i(x) \leq 0$, $i \in \mathcal{I}$ and $c_i(x) = 0$, $i \in \mathcal{E}$ are called inequality constraints and equality constraints. These definitions allow us to define the general constrained optimization problem:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & \begin{array}{l} c_i(x) \leq 0 \quad i \in \mathcal{I} \\ c_i(x) = 0 \quad i \in \mathcal{E}. \end{array} \end{array} \tag{1.3}$$

For simplicity of notations, the problem (1.3) is sometimes written

$$\min_{x \in \Omega} f(x), \tag{1.4}$$

where $\Omega \subset \mathbb{R}^n$ is the feasible region.

The formulation (1.3) is quite general. For many years researchers have constructed many methods for some subclasses of optimization problems where for example the objective function and the constraints are linear, or where the variables are not continuous but discrete, ... It is beyond the scope of this thesis to present all these different classes but the interested reader may consult Fletcher (1987), Gill, Murray and Wright (1981) or Nocedal and Wright (1999).

The presence of constraints on the variables is clearly important in the solution of optimization problems. In this thesis we only consider unconstrained optimization problems or bound-constrained optimization problems. In this last class, the only constraints have the following form:

$$l_i \leq x_i \leq u_i \quad i = 1, \ldots, n, \tag{1.5}$$

where l_i and $u_i \in \mathbb{R}$ are lower and upper bounds on the i-th component of $x \in \mathbb{R}^n$. Note that any of the bounds may be infinite. Without loss of generality, we assume throughout this work that $l_i < u_i$ for all $i = 1, \ldots, n$. This class of problems is widely studied since many softwares solve general constrained optimization problems by solving a sequence of bound-constrained optimization problems.

If the objective function is nonlinear, the associated optimization problem is called *nonlinear optimization problem*. This is an important class of problems since nonlinearity is nearly unavoidable in many real-world problems. This thesis addresses the solution of this class of problems.

1.2 Notions and Notations

Before introducing nonlinear optimization more in details, we introduce in this section the basic notions and notations used in this thesis.

1.2.1 Basic notions

A complete description of the notions presented in this section may be found in Chapter 2 of Golub and Van Loan (1989). Let us denote by $\mathbb{R}^n$ the Euclidean space of dimension n. A

vector is a n-dimensional column of the form

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}, \tag{1.6}$$

where each x_i $(i = 1, \ldots, n)$ is called the i-th *component* of the vector x. We denote by x^T, the *transpose* of the vector x, that is a n-dimensional row of the form

$$x^T = \begin{pmatrix} x_1 & x_2 & \ldots & x_{n-1} & x_n \end{pmatrix}. \tag{1.7}$$

If there is no confusion, we denote by x_k, either the k-th component of vector x or a vector that is the k-th iteration of an iterative algorithm.

We denote the *inner product* of two vectors x and y by

$$\langle x, y \rangle \stackrel{\text{def}}{=} x^T y = \sum_{i=1}^{n} x_i y_i. \tag{1.8}$$

We denote by A_{ij} the component (i, j) of matrix A. We denote A^T the *transpose* of matrix A, that is the matrix B where each component B_{ij} is the component (j, i) of matrix A. A matrix A is said to be *symmetric* if $A^T = A$. A symmetric matrix A is said *positive semidefinite* if

$$x^T A x \geq 0 \quad \forall x \in \mathbb{R}^n. \tag{1.9}$$

It is said *positive definite* if

$$x^T A x > 0 \quad \forall x \in \mathbb{R}^n, \quad x \neq 0. \tag{1.10}$$

And the matrix is said *indefinite* if there exists two vectors $x, y \in \mathbb{R}^n$ such that

$$x^T A x > 0 \quad \text{and} \quad y^T A y < 0. \tag{1.11}$$

1.2.2 Norms

A *norm* on the Euclidean space $\mathbb{R}^n$ is a function $\| \cdot \| : \mathbb{R}^n \to \mathbb{R}$ which has the following properties

positive homogeneity	$\forall a \in \mathbb{R}, \forall x \in \mathbb{R}^n, \|ax\| = \|a\| \, \|x\|$.
triangle inequality	$\forall x, y \in \mathbb{R}^n, \|x + y\| \leq \|x\| + \|y\|$.
positive definiteness	$\|x\| = 0 \Leftrightarrow x = 0$.

A commonly-used norm is called the *Euclidean norm* (or 2-norm) and is defined by:

$$\|x\|_2 \stackrel{\text{def}}{=} \sqrt{x^T x} = \sqrt{\sum_{i=1}^{n} x_i^2}. \tag{1.12}$$

This norm is a particular case of the p-norm which is defined by

$$\|x\|_p \stackrel{\text{def}}{=} \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \tag{1.13}$$

where $p \geq 1$. Other particular cases of the p-norm are given by

1-norm $\|x\|_1 \stackrel{\text{def}}{=} \sum_{i=1}^{n} |x_i|$
∞**-norm** $\|x\|_\infty \stackrel{\text{def}}{=} \max_i |x_i|$

Two norms are said *equivalent* if the first could be bounded up and down by the other and vice versa. In $\mathbb{R}^n$, all norms are equivalent. The equivalence relations for the p-norms are, $\forall x \in \mathbb{R}^n$,

$$\begin{array}{rcccl} \|x\|_2 & \leq & \|x\|_1 & \leq & \sqrt{n}\|x\|_2 \\ \|x\|_\infty & \leq & \|x\|_2 & \leq & \sqrt{n}\|x\|_\infty \\ \|x\|_\infty & \leq & \|x\|_1 & \leq & n\|x\|_\infty. \end{array} \tag{1.14}$$

Note also that the Euclidean norm satisfies the *Cauchy-Schwarz inequality*, that is

$$|x^T y| \leq \|x\| \, \|y\|. \tag{1.15}$$

One can also define norms on matrix spaces. A *matrix-norm*, denoted by $\|A\|$, is a function $\| \cdot \| : \mathbb{R}^{m \times n} \to \mathbb{R}$ which has the following properties

$\forall a \in \mathbb{R}, \forall A \in \mathbb{R}^{m \times n}$, $\|aA\| = |a| \, \|A\|$.

$\forall A, B \in \mathbb{R}^{m \times n}$, $\|A + B\| \leq \|A\| + \|B\|$.

$\|A\| \geq 0$ and $\|A\| = 0 \Leftrightarrow A = 0$.

$\forall A, B \in \mathbb{R}^{m \times n}$, $\|AB\| \leq \|A\|\|B\|$.

Some well-known matrix norms are the induced p-norms defined by

$$\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}. \tag{1.16}$$

Some particular cases are given by

1-norm $\|A\|_1 \stackrel{\text{def}}{=} \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$,

2-norm $\|A\|_2 \stackrel{\text{def}}{=} \lambda_{max}(A^T A)$ where $\lambda_{max}(\cdot)$ denotes the largest eigenvalue,

∞**-norm** $\|A\|_\infty \stackrel{\text{def}}{=} \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|$.

Another well-known norm used in optimization is the Frobenius norm defined by

$$\|A\|_F \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |A_{ij}|^2}. \tag{1.17}$$

1.2.3 Elements of topology

We now present some elements of topology commonly used in optimization (see Sutherland (1975) for a good introduction to topology elements). A *neighborhood* $\mathcal{N}$ of the point $x \in \mathbb{R}^n$ is an open set containing x. A commonly used neighborhood is the ball of radius ϵ centered at x

$$\mathcal{B}_\epsilon(x) \stackrel{\text{def}}{=} \{y \in \mathbb{R}^n \mid \|x - y\| < \epsilon\}. \tag{1.18}$$

A subset $\mathcal{S}$ of $\mathbb{R}^n$ is said to be *open* if for each vector $x \in \mathcal{S}$ there exists a constant $\epsilon > 0$ such that $\mathcal{B}_\epsilon(x) \subset \mathcal{S}$. We say that a set $\mathcal{S}$ is *closed* if and only if its complement in $\mathbb{R}^n$, that is $\mathbb{R}^n \backslash \mathcal{S}$, is open. A subset $\mathcal{S}$ of $\mathbb{R}^n$ is said to be *bounded* if there exists a constant $\kappa > 0$ such that

$$\|x\| \leq \kappa \quad \forall x \in \mathcal{S}. \tag{1.19}$$

Finally, a set $\mathcal{S}$ of $\mathbb{R}^n$ is *compact* if and only if it is both closed and bounded. Note that this last definition derives by the Heine-Borel theorem from the general definition of compact spaces and is only valid for subsets of Euclidean space.

1.2.4 Function types

We now present two function types which are commonly used in optimization (for more details, see Luenberger (1969) or Rockafellar (1970)). A function f, defined on a convex set $\mathcal{S}$, is *convex* if for all $x, y \in \mathcal{S}$,

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y) \quad \forall \lambda \in [0, 1]. \tag{1.20}$$

A function f is called *strictly convex* if, for $x \neq y$ and $\lambda \in (0, 1)$, the inequality (1.20) is strict. We say that a function f is *concave* if $(-f)$ is convex.

Let $\mathcal{S}$ be an open subset of $\mathbb{R}^n$. The function $f : \mathcal{S} \to \mathbb{R}^m$ is said to be *Lipschitz continuous* on $\mathcal{S}$ if there exists a constant $M > 0$ such that for all $x, y \in \mathcal{S}$,

$$\|f(x) - f(y)\| \leq M\|x - y\|. \tag{1.21}$$

1.2.5 Convergence

We now present the notions of convergence. This is an important notion to qualify an iterative optimization algorithm. A sequence $\{x_k\}_{k \in \mathbb{N}}$, where $x_k \in \mathbb{R}^n \, \forall k$, is said to be *convergent* to x^* if

$$\forall \epsilon > 0, \, \exists K \in \mathbb{N} : \forall k \geq K \; \|x_k - x^*\| < \epsilon, \tag{1.22}$$

and we denote it by

$$x_k \to x^* \quad \text{or} \quad \lim_{k \to \infty} x_k = x^*. \tag{1.23}$$

A sequence $\{x_k\}$ is called a *bounded sequence* if there exists a constant $\kappa > 0$ such that $\|x_k\| \leq \kappa$ for all k. We say that x^* is a *limit point* or an *accumulation point* of the sequence $\{x_k\}$ if there exists a subsequence $\{x_k\}_{k \in K} \subset \{x_k\}$ such that $\{x_k\}_{k \in K}$ converges to x^*.

In optimization, a very important concept is the rate of convergence. Indeed, the speed of convergence of an infinite sequence of iterates $\{x_k\}$ is a good way to compare algorithms. One of the most efficient ways to assess the rate of convergence is to compare the progress at a step with the progress at the previous step, this is called the Q-rate. We say that a sequence $\{x_k\}$ converges *Q-linearly* to x^* if

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = \mu \quad \text{with } 0 < \mu < 1. \tag{1.24}$$

If (1.24) holds with the rate of convergence μ equal to zero, one says that the sequence converges *Q-superlinearly*. More generally, a sequence $\{x_k\}$ is said to be convergent with Q-order r for $r > 1$ to x^* if

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^r} = \mu \quad \text{with } \mu > 0. \tag{1.25}$$

In particular, convergence with Q-order 2 is called *Q-quadratic* convergence. For clarity, we will here omit the letter Q and simply talk about linear convergence, quadratic convergence, ... A complete description of the rates of convergence may be found in Ortega and Rheinboldt (1970).

In optimization, two types of convergence are also of practical importance. The first is the global convergence. Assume that a sequence $\{x_k\}$ is generated by an algorithm, this latter is said to be *globally convergent* when $\{x_k\}$ has an accumulation point for any initial iterate. On the contrary, if the sequence only converges for initial iterates close enough to an accumulation point, the convergence is said to be *local*.

1.2.6 Derivatives

Differentiability is very important in optimization because most algorithms use available information about a function at one point to deduce its behavior at other points. If the problem derivatives are available, the capability of an algorithm to find a solution is strengthened compared with problems without derivatives. The interested reader may see Gruver and Sachs (1980) and Dennis and Schnabel (1983) for more details about this section.

Let us consider a function $f : \mathbb{R}^n \to \mathbb{R}$. The *first partial derivative* of f with respect to the variable x_i is defined by the following limit (if it exists) :

$$\frac{\partial f(x)}{\partial x_i} = \lim_{\epsilon\to 0} \frac{f(x + \epsilon e_i) - f(x)}{\epsilon}, \tag{1.26}$$

where e_i is the i-th unit vector. Assuming that all of these partial derivatives exist, the gradient of f at x is defined as the n-dimensional vector

$$\nabla_x f(x) = g(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix}. \tag{1.27}$$

the function f is said to be *differentiable at* $x_0 \in \mathbb{R}^n$ if

$$\lim_{x\to x_0} \frac{f(x) - f(x_0) - \nabla_x f(x_0)^T (x - x_0)}{\|x - x_0\|} = 0. \tag{1.28}$$

If the function f is differentiable for all $x \in \mathbb{R}^n$, f is said *differentiable*. Note that if f is differentiable at x, f is continuous at x. If the derivatives are further continuous functions of x, f is said to be *continuously differentiable*.

If the second partial derivatives of a function f defined by

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\frac{\partial f(x)}{\partial x_j} \tag{1.29}$$

exist for all $i, j (1 \leq i, j \leq n)$ and are continuous functions of x, then f is said to be *twice-continuously differentiable*, that is $f \in C^2$. These n^2 second partial derivatives are represented by an n-by-n, symmetric matrix known as the *Hessian matrix* of f or simply the Hessian :

$$\nabla^2_{xx} f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial {x_1}^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial {x_2}^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial {x_n}^2} \end{pmatrix}. \tag{1.30}$$

One of the results from analysis that is most frequently used in optimization theory is the following theorem.

Theorem 1.2.1 (Mean value theorem) Assume that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and that $s \in \mathbb{R}^n$. Then,

$$f(x+s) = f(x) + \nabla_x f(x+\alpha s)^T s, \tag{1.31}$$

for some $\alpha \in (0,1)$. If, in addition, f is twice-continuously differentiable, then

$$\nabla_x f(x+s) = \nabla_x f(x) + \int_0^1 \nabla^2_{xx} f(x+\alpha s) s d\alpha, \tag{1.32}$$

and

$$f(x+s) = f(x) + \nabla_x f(x)^T s + \tfrac{1}{2} s^T \nabla^2_{xx} f(x+\alpha s) s \tag{1.33}$$

for some $\alpha \in (0,1)$.

This theorem shows that, if the function and its first and second derivatives are known at a point x, then we can build approximations to that function at all points in the neighborhood of x. In particular, we may approximate $f(x+s)$ by its first-order Taylor approximation

$$f(x+s) \approx f(x) + \nabla_x f(x)^T s, \tag{1.34}$$

or by its second-order Taylor approximation

$$f(x+s) \approx f(x) + \nabla_x f(x)^T s + \tfrac{1}{2} s^T \nabla^2_{xx} f(x) s \tag{1.35}$$

Note that equations (1.34) and (1.35) are also respectively called *linear* and *quadratic models* of the function f around the point x and are widely used in algorithms designed for solving optimization problems.

1.3 Nonlinear optimization

This section is devoted to a particular class of optimization problems, that is nonlinear problems (NLP). The research that we have developed during our thesis is entirely devoted to this class of problems. This is a subject of practical importance since the applications from industry become more and more complicated and thus nonlinearity is clearly unavoidable.

1.3.1 Characterization of solutions

Let us remind the general mathematical program

$$\min_{x \in \mathcal{S}} f(x), \tag{1.36}$$

where f is a continuously differentiable function from $\mathbb{R}^n$ into $\mathbb{R}$ and the feasible set $\mathcal{S}$, is a subset of $\mathbb{R}^n$.

We first characterize the different possible solutions of this class of problems. A point x^* is a *global minimizer* of f if

$$f(x^*) \leq f(x) \quad \text{for all } x \in \mathcal{S}. \tag{1.37}$$

In nonlinear optimization, finding the global minimizer of a problem is very complicated and we thus only aim at finding a local minimizer, which is a point that has the lowest value of f in its neighborhood. More formally, we say that a point x^* is a *(weak) local minimizer* if there is a neighborhood $\mathcal{N}$ of x^* such that

$$f(x^*) \leq f(x) \quad \text{for all } x \in \mathcal{N} \cap \mathcal{S}. \tag{1.38}$$

A *strict local minimizer* is a point x^* if there exists a neighborhood $\mathcal{N}$ of x^* such that

$$f(x^*) < f(x) \quad \text{for all } x \in \mathcal{N} \cap \mathcal{S} \text{ with } x \neq x^*. \tag{1.39}$$

We call *minimum* the value of the objective function f at a minimizer. The figure 1.1 illustrates the types of minimizers defined above.

Note that, in this work, we restrict our attention to the computation of local minimizers. The interested reader may see Fiacco and McCormick (1968), Mangasarian (1979), Gill et al. (1981) and Fletcher (1987) for more details about this section.

1.3.2 Optimality conditions

The definition of the different minimizers stated in the previous section is quite abstract. This is why mathematicians have written conditions which are simpler to apply to a point to know if it is or not an optimal solution of the optimization problem. In the two following sections, we state the classical optimality conditions for general unconstrained and bound-constrained optimization problems. Further details about optimality conditions may be found, for instance, in Conn, Gould and Toint (2000), Nocedal and Wright (1999), Fiacco and McCormick (1968), Mangasarian (1979), Gill et al. (1981) and Fletcher (1987).

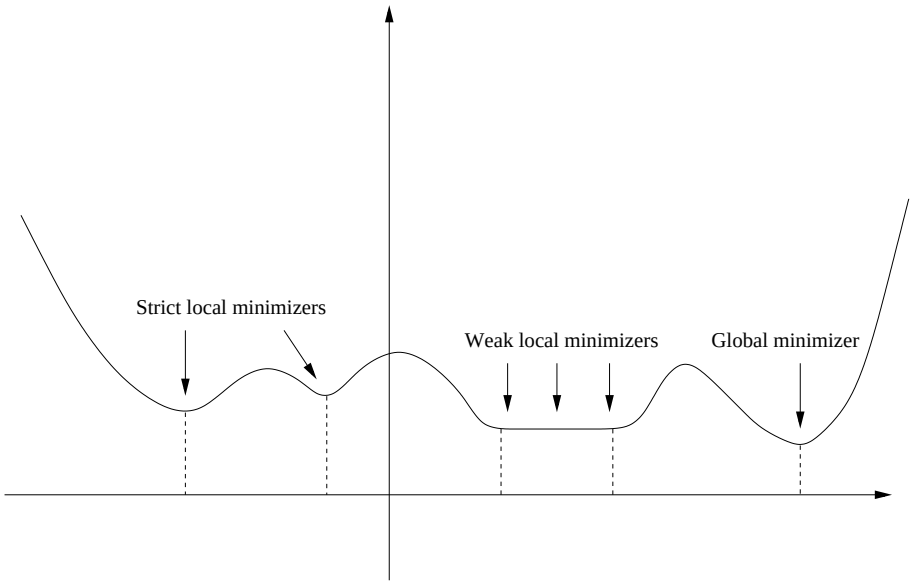

Figure 1.1: Examples of local and global minimizers in one dimension.

1.3.2.1 Unconstrained optimization

Let us consider the unconstrained minimization problem

$$\min_{x \in \mathbb{R}^n} f(x). \tag{1.40}$$

We begin with necessary conditions for optimality. These are deduced by assuming that a point x^* is a local minimizer and then establishing some properties of the first-order and second-order derivatives.

Theorem 1.3.1 (First-order necessary condition) Suppose that x^* is a local minimizer of problem (1.40) and f is continuously differentiable in a neighborhood of x^* . Then

$$\nabla_x f(x^*) = 0. \tag{1.41}$$

A point x^* satisfying (1.41) is referred to as a *first-order critical* or *first-order stationary point* of f . Note that any local minimizer must be a first-order critical point.

Theorem 1.3.2 (Second-order necessary conditions) Suppose that x^* is a local minimizer of problem (1.40) and f is twice-continuously differentiable in a neighborhood of x^*. Then

$$\nabla_x f(x^*) = 0 \text{ and } \nabla_{xx} f(x^*) \text{ is positive semidefinite.} \tag{1.42}$$

In this case, we say that x^* is a *second-order critical point*. The following theorem states under what conditions a point x^* is a local minimizer of the objective function f .

Theorem 1.3.3 (Second-order sufficient conditions) Suppose that f is twice-continuously differentiable in a neighborhood of x^* and that furthermore

$$\nabla_x f(x^*) = 0 \text{ and } \nabla_{xx} f(x^*) \text{ is positive definite.} \tag{1.43}$$

Then x^* is a strict local minimizer of problem (1.40).

Note that, if x^* is first-order critical but the Hessian is indefinite, we say that x^* is a *saddle point*. If the objective function f is convex, optimality conditions are simpler. Indeed, every local minimizer of this function is also a global minimizer (see Luenberger (1969) or Rockafellar (1970)). Finally, we point out the fact that optimality conditions often provide the foundations for the development and the analysis of iterative algorithms. In the case of unconstrained optimization, algorithms search for points at which the gradient of f vanishes. In practice, algorithms terminate when these optimality conditions hold approximately.

1.3.2.2 Bound-constrained optimization

Let us now consider the bound-constrained optimization problem

$$\begin{array}{ll} \min & f(x), \\ \text{s.t.} & l \leq x \leq u, \end{array} \tag{1.44}$$

where f is a twice continuously differentiable function of the variables $x \in \mathbb{R}^n$ and l and $u \in \mathbb{R}^n$ represent lower and upper bounds on the variables. Note that any of these bounds may be infinite. Without loss of generality, we assume that $l_i < u_i$ for all $i = 1, \ldots, n$.

The set of points which satisfy the constraints in problem (1.44) is the feasible box and is denoted by

$$\Omega = \{x \in \mathbb{R}^n \,|\, l \leq x \leq u\}. \tag{1.45}$$

Any point belonging to this box is said to be *feasible*. An *active constraint* indicates that a variable lies on one of its bounds. The *active set* is then defined as the following set

$$\mathcal{A}(x) \stackrel{\text{def}}{=} \{i \,|\, x_i = l_i \text{ or } x_i = u_i\}. \tag{1.46}$$

The first-order necessary condition can be expressed as the following theorem.

Theorem 1.3.4 (First-order necessary condition for a bound-constrained problem) Suppose that x^* is a local minimizer of problem (1.44) and f is continuously differentiable in a neighborhood of x^*. Then, defining the *binding set* as

$$\mathcal{B}(x^*) \stackrel{\text{def}}{=} \left\{ i \,|\, x_i^* = l_i \text{ and } \frac{\partial f(x^*)}{\partial x_i} \geq 0 \right\} \cup \left\{ i \,|\, x_i^* = u_i \text{ and } \frac{\partial f(x^*)}{\partial x_i} \leq 0 \right\} \tag{1.47}$$

we have

$$\frac{\partial f(x^*)}{\partial x_i} = 0,\ i \notin \mathcal{B}(x^*). \tag{1.48}$$

The latter theorem essentially requires that all partial derivatives of f with respect to x_i which are not at their upper or lower bounds be zero, and those partial derivatives with respect to x_i which are at a bound must be larger than zero at the lower bound and less than zero at the upper one. Second-order sufficient conditions for x^* to be a local minimizer of problem (1.44) are stated as follows.

Theorem 1.3.5 (Second-order sufficient conditions for a bound-constrained problem) Suppose that the first-order condition (1.48) holds and that, furthermore,

$$s^T \nabla_{xx} f(x^*) s > 0 \text{ for all vectors } s,\ s \neq 0,\ s_i = 0 \quad i \in \mathcal{B}_s(x^*), \tag{1.49}$$

where $\mathcal{B}_s(x^*)$ is known as the *strictly binding set* at x^* and is defined as

$$\mathcal{B}_s(x^*) \stackrel{\text{def}}{=} \mathcal{B}(x^*) \cap \left\{ i \,|\, \frac{\partial f(x^*)}{\partial x_i} \neq 0 \right\}. \tag{1.50}$$

Then x^* is a local minimizer of problem (1.44).

Considering the set of indexes of free variables (variables which are not at one of their bounds), which is of the form

$$\mathcal{F}(x) \stackrel{\text{def}}{=} \{ i \,|\, l_i < x_i < u_i \}, \tag{1.51}$$

the restricted gradient and the restricted Hessian are, respectively, the gradient and the Hessian of the objective function with respect to free variables, i.e. to $x_i (i \in \mathcal{F}(x))$. So algorithms designed for the solution of box-constrained optimization problems typically perform by identifying these free variables and then using unconstrained minimization methods to explore the corresponding "restricted" problem in order to drive the restricted gradient to zero.

1.3.3 Criticality measures

In order to check if the optimality conditions are satisfied, we need a measure of the criticality of the solution. We define $\pi(k, x_k)$ to be a *first-order criticality measure* of the iterate x_k if it is a nonnegative real function of its second argument such that

$$\|x_k - x_l\| \to 0 \text{ implies that } |\pi(k, x_k) - \pi(l, x_l)| \to 0 \tag{1.52}$$

and if the limit

$$\lim_{k\to\infty} \pi(k, x_k) = 0 \tag{1.53}$$

corresponds to asymptotically satisfying the first-order criticality conditions of the optimization problem under study.

In unconstrained optimization, the most used criticality measure is the gradient norm (the Euclidean or the infinity norm). But in the case of bound-constrained optimization, the situation is more complicated. Let us define the *projection operator*, denoted by $P[\cdot, l, u]$, defined componentwise by

$$P[x, l, u]_i \stackrel{\text{def}}{=} \begin{cases} l_i & \text{if} \quad x_i \leq l_i, \\ x_i & \text{if} \quad l_i < x_i < u_i, \\ u_i & \text{if} \quad x_i \geq u_i. \end{cases} \tag{1.54}$$

One can define the *projected-gradient path*

$$x_k(t) \stackrel{\text{def}}{=} P[x_k - t\nabla_x f(x_k), l, u] \quad t \geq 0, \tag{1.55}$$

which is simply the projection of the *gradient path* $x_k - t\nabla_x f(x_k)\,(t \geq 0)$ on the feasible box as shown in Figure 1.2. One can then define the *projected gradient* of the objective function f on the feasible box as a part of this path

$$\bar{g}(x) \stackrel{\text{def}}{=} x - P[x - \nabla_x f(x), l, u]. \tag{1.56}$$

A common criticality measure is then

$$\pi(k, x_k) = \pi(x_k) \stackrel{\text{def}}{=} \|x_k - P[x_k - \nabla_x f(x_k), l, u]\|_\infty = \|\bar{g}(x_k)\|_\infty. \tag{1.57}$$

Another commonly used criticality measure that we will sometimes prefer in this thesis is defined by

$$\chi(x_k) \stackrel{\text{def}}{=} | \min_{\substack{x_{i,k} + d \in \mathcal{F} \\ \|d\|_\infty \leq 1}} \langle \nabla_x f(x_k), d \rangle |, \tag{1.58}$$

which can be interpreted as the magnitude of the maximum decrease of the linearized model of the objective function f achievable on the intersection of the feasible domain with a ball of radius 1 centered at x_k (see Conn, Gould, Sartenaer and Toint, 1993, for a full explanation of this measure). Note that this criticality measure simply reduces to the gradient norm for unconstrained problems. We also point out the recent thesis of Melodie Mouffe (Mouffe (2009)) which contains a comparative analysis of the different criticality measures from the backward-error point of view.

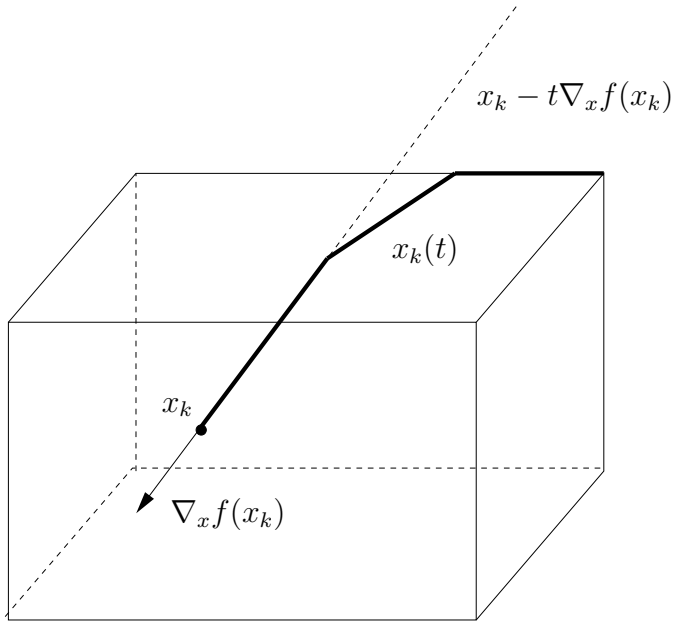

Figure 1.2: A projected-gradient path $x_k(t)$ in $\mathbb{R}^3$.

1.3.4 Nonlinear methods

We now present a review of the methods used to solve nonlinear optimization problems. These methods commonly iteratively produce a sequence of iterates $\{x_k\}$ hoping that it converges to a solution of the problem.

1.3.4.1 Newton's method

Assuming a convex quadratic function is given, the most known technique to solve unconstrained optimization problems is Newton's method (see Ortega and Rheinboldt (1970) for a thorough description of Newton's method and its convergence properties), which is an iterative one and whose aim is to solve the equation

$$\nabla_x f(x) = 0. \tag{1.59}$$

Let us consider the quadratic model of f at the current iterate x_k

$$m_k(x_k + s) = f(x_k) + \nabla_x f(x_k)^T s + \tfrac{1}{2} s^T \nabla^2_{xx} f(x_k) s \tag{1.60}$$

The idea of the method is to minimize the model at the current iterate. If the Hessian $\nabla^2_{xx} f(x_k)$ is positive definite, then the quadratic model (1.60) has a unique minimizer s_k at a point where the gradient of this model vanishes, i.e. where

$$\nabla_x f(x_k) + \nabla^2_{xx} f(x_k) s = 0. \tag{1.61}$$

As long as $\nabla^2_{xx} f(x_k)$ remains positive definite for all x, Newton's method is well defined. It is summarized by the algorithm 1.3.1.

Algorithm 1.3.1: Newton's method

Step 0. An initial point $x_0 \in \mathbb{R}^n$ is given. Set $k = 0$.

Step 1. Compute the step s_k by solving the system of linear equations

$$\nabla^2_{xx} f(x_k) s_k = -\nabla_x f(x_k). \tag{1.62}$$

Step 2. Set

$$x_{k+1} = x_k + s_k. \tag{1.63}$$

Increment k by one and go to Step1.

Equations (1.62) are called *Newton equations* and the solution s_k is known as the *Newton step* or *Newton direction*. If the Hessian is positive definite, it then follows from (1.62) that

$$\nabla_x f(x_k)^T s = -s^T \nabla^2_{xx} f(x_k) s_k \leq -\sigma_k \|s_k\|^2 \quad \text{for some } \sigma_k > 0. \tag{1.64}$$

Therefore, if the gradient of f is nonzero, the Newton direction is a descent direction, i.e., $s^T \nabla_x f(x_k) < 0$. If the gradient is zero, then the step s_k is also zero. However, far from a local minimum, the Hessian matrix may be singular or the Newton direction may not be a descent direction if $\nabla^2_{xx} f(x_k)$ is not positive definite.

Another important characteristic of Newton's method is that, when it works, it converges rapidly; the asymptotic rate of convergence is quadratic. Note the fact that Newton's method converges after one iteration for all quadratic objective functions. However, one of its most serious disadvantages is the lack of global convergence. As the Taylor's approximation (1.60) is only valid in the proximity of the solution x^* , Newton's method is suitable only when the initial point is close enough to x^* . The requirement of analytic second-order derivatives of the objective function is another drawback to Newton's method. These derivatives may be difficult to determine notwithstanding the fact that they are known to exist. Mathematicians have derived methods, called quasi-Newton methods, that do not necessitate to compute the exact Hessian (see Perry (1976), Shanno (1978) or Nocedal (1980)). Instead, they use an approximation which is updated at each iteration. Some well-known approximations are for instance the BFGS formula (which was independently proposed by Broyden (1970), Fletcher (1970), Goldfarb (1970) and Shanno (1970)) or the SR1 technique (which was first proposed by Davidon (1968)). Finally, from our point of view, the advantages and disadvantages of Newton's method are the foundations to the development and improvement of more practical algorithms.

To overcome the disadvantages of Newton's method, people have constructed globalization techniques. These approaches are applied in order to enforce algorithms to converge from any initial point. When we are far away from a solution, these globalization techniques are an active part of the process because they avoid displacement away from the solution or even divergence. But close to an optimum, they will play the role of safeguards; they may be used if required, but usually they will not be invoked. We present in this section two globalization techniques, the

well-known linesearch methods and the recent adaptive cubic overestimation technique. Since we particularly focus on the third globalization technique, namely the trust-region method, we will describe it more in details in the next section.

1.3.4.2 Linesearch methods

We start our survey of globalization techniques with linesearch methods. After having computed a direction d_k , each iteration of a linesearch method must decide how long the step in this direction should be. The iterates are generated by

$$x_{k+1} = x_k + \alpha_k d_k, \tag{1.65}$$

where $d_k \in \mathbb{R}^n$ is the direction and $\alpha_k > 0$ is known as the step length, computed by an appropriate linesearch method, whose goal is to sufficiently reduce the value of the objective function f while taking sufficiently large steps. The best choice for the step length α_k is theoretically the solution of the following minimization problem

$$\min_{\alpha>0} f(x_k + \alpha d_k). \tag{1.66}$$

However, the exact minimizer of (1.66) is often expensive to compute and unnecessary. Therefore, instead of solving (1.66) exactly and performing an exact linesearch, it is generally preferable to solve this problem only approximately: this process is known as inexact linesearch. There are several rules for choosing the step length α. In practice, one solves this problem approximately by imposing some conditions ensuring a sufficient decrease, as, for instance, Armijo conditions or Wolfe conditions (see Nocedal and Wright (1999)). Note that, in order to benefit from the properties of fast convergence of Newton-type methods, we always try the unit step length $\alpha = 1$ first. If it is not possible, a backtracking procedure is performed. The value resulting from this procedure is generated by moving backwards from $\alpha = 1$ to $\alpha = 0$.

1.3.4.3 Adaptive cubic overestimation technique

Recently, Cartis, Gould and Toint (2009) have proposed an other globalization method called adaptive cubic overestimation (ACO). Actually, this type of methods was first initiated by Griewank (1981), Nesterov and Polyak (2006) and Weiser, Deuflhard and Erdmann (2007) and have been consolidated into a practical and successful algorithm. At each iteration of this method, we generate the next iterate by

$$x_{k+1} = x_k + s_k, \tag{1.67}$$

where s_k is the solution of the minimization of a third-order model of the objective function f defined by

$$m_k(x_k + s) \stackrel{\text{def}}{=} f(x_k) + s^T \nabla_x f(x_k) + \tfrac{1}{2} s^T B_k s + \tfrac{1}{3}\sigma_k \|s\|_2^3, \tag{1.68}$$

where B_k is a symmetric approximation of the objective Hessian $H(x_k) = \nabla_{xx} f(x_k)$. This step is only accepted if the adequation between the objective function and the model is large

enough. The third-order parameter σ_k is then updated in order that the model fit the objective function as well as possible.

The step done at each iteration is done by either approximately or exactly minimizing the model given in (1.68). Good convergence and complexity properties for this algorithm have been derived and preliminary numerical experiments on all the unconstrained problems of the CUTEr test set (see Gould, Orban and Toint (2003*a*)) have shown that it compares well with a trust-region algorithm.

1.4 A basic trust-region algorithm (BTR)

We now consider another class of globalization techniques, called trust regions, which is the approach we consider in our research work to promote global convergence. The first methods that we might consider as trust-region ones are due to Levenberg (1944) and Marquardt (1963) and are used to solve nonlinear least-squares problems. The method initiated was then developed further by Morrison (1960). In Powell (1970) the trust-region concept is advocated as a tool to ensure convergence of a method for unconstrained optimization.

The basic idea of trust-region methods is to accept the minimum of a quadratic model only as long as the latter adequately reflects the behavior of the objective function f. In contrast to linesearch techniques, trust-region methods do not require the positive definiteness of the Hessian of the quadratic model. Another interesting feature of trust-region methods is that they have the possibility to naturally take advantage of directions of negative curvature when they are present (for instance when the Hessian is indefinite and the local quadratic model unbounded below). Indeed, these directions may be taken safely to the boundary of the trust region.

1.4.1 The method

At each iteration k, trust-region methods build a model of the objective function f around the current iterate x_k. The most practical choice for this model is a quadratic one of the form

$$m_k(x_k + s) = f(x_k) + g_k^T s + \tfrac{1}{2} s^T H_k s, \tag{1.69}$$

where $g_k \stackrel{\text{def}}{=} \nabla_x m_k(x_k) = \nabla_x f(x_k)$, and H_k is either the objective function's Hessian $\nabla_{xx} f(x_k)$ or some symmetric approximation of it. Trust-region methods then minimizes the model in a neighborhood of the iterate x_k, in which we believe the model to be sufficiently adequate. This neighborhood is called *trust region* and is defined as,

$$\mathcal{B}_k = \{x_k + s \mid \|s\|_k \leq \Delta_k\}, \tag{1.70}$$

where $\Delta_k > 0$ is known as the *trust-region radius* and $\|\cdot\|_k$ is an iteration-dependent norm.

Using the trust region, we define the following subproblem

$$\begin{array}{ll} \min & m_k(x_k + s) \\ \text{s.t.} & \|s\|_k \leq \Delta_k, \end{array} \tag{1.71}$$

which is called the *trust-region subproblem*. A *trial step* s_k is then computed by possibly approximately minimizing (1.71). Let us denote the *trial point* by $x_k^+ = x_k + s_k$. Trust-region algorithms then evaluate the objective function at the candidate point and accept x_k^+ as the new iterate if the reduction achieved in the objective function is at least a given fraction of that predicted by the model. The trust-region radius Δ_k is also possibly enlarged if this ratio is sufficiently large and the iteration is declared to be *successful*. The increase may indicate that longer steps would be successful as well. Otherwise, if the achieved reduction is too small, the trial point is rejected, the trust-region radius is reduced and the iteration is said to be *unsuccessful*. We then conclude that the model is not sufficiently accurate in this trust region.

Formally, we compute the ratio between the *achieved reduction* (i.e. the decrease in f) versus the *predicted reduction* (i.e. the reduction in m_k)

$$\rho_k \stackrel{\text{def}}{=} \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}. \tag{1.72}$$

If this ratio is close to one, it means that the model approximates the objective function well.

The detailed trust-region algorithm, which is one of the basic components of methods developed in this research work, is outlined in Algorithm 1.4.1.

Algorithm 1.4.1: Basic Trust-Region (BTR) algorithm

Step 0: Initialization. An initial point x_0 and an initial trust-region radius $\Delta_0 > 0$ are given. The constants η_1, η_2, γ_1 and γ_2 also given and satisfy

$$0 < \eta_1 \leq \eta_2 < 1 \quad \text{and} \quad 0 < \gamma_1 \leq \gamma_2 < 1. \tag{1.73}$$

Compute $f(x_0)$ and set the iteration counter k to 0.

Step 1: Model definition. Compute the model m_k approximating f around x_k in $\mathcal{B}_k$.

Step 2: Step calculation. Compute a step s_k that "sufficiently reduces" the model m_k and such that $x_k + s_k \in \mathcal{B}_k$.

Step 3: Acceptance of the trial point. Compute $f(x_k + s_k)$ and define ρ_k as in (1.72). If $\rho_k \geq \eta_1$, set $x_{k+1} = x_k + s_k$; otherwise define $x_{k+1} = x_k$.

Step 4: Trust-region radius update. Update the radius as follows

$$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty) & \text{if } \rho_k \geq \eta_2, \\ [\gamma_2 \Delta_k, \Delta_k] & \text{if } \rho_k \in [\eta_1, \eta_2), \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \text{if } \rho_k < \eta_1, \end{cases} \tag{1.74}$$

set $k = k + 1$ and go to Step 1.

One of the critical tasks in such an algorithm is the computation of the trial step s_k which "sufficiently reduces" the model within the current trust region. This sufficient decrease may be measured in terms of the Cauchy point. Consider the *Cauchy arc*

$$x_k^C(t) \stackrel{\text{def}}{=} \{x \,|\, x = x_k - t\nabla_x f(x_k),\ t \geq 0 \text{ and } x \in \mathcal{B}_k\}. \tag{1.75}$$

If our model is a quadratic one of the form (1.69), then it is possible to calculate the exact minimum of the model m_k along this arc. The computing minimizer x_k^C is known as the *Cauchy point* (see Dennis (1978)). The Cauchy point is of essential importance in the convergence analysis of algorithms based on trust-region methods. Such methods are proved to be globally convergent if steps s_k give a reduction in the model that is at least a fraction of the decrease obtained at the Cauchy point. More formally, the condition on the reduction in the model is given by

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa(m_k(x_k) - m_k(x_k^C)), \tag{1.76}$$

for some constant $\kappa \in (0, 1)$.

1.4.2 The solution of the trust-region subproblem

There exist many methods to solve the trust-region subproblem either exactly or approximately. We present here two different methods. The first, the Moré-Sorensen technique is used to solve exactly the trust-region subproblem but can be expensive due to the need to factorize the Hessian matrix and it is thus often unaffordable to solve large scale problems (typically when the Hessian factorization is itself expensive due to fill-in). The second method presented is the truncated conjugate gradient method and extends the conjugate gradient method for the solution of linear systems to solve quadratic minimization problems.

1.4.2.1 The Moré-Sorensen algorithm

Let us assume that we use an Euclidean norm for the trust-region algorithm. Let us consider the unconstrained trust-region subproblem

$$\begin{array}{ll} \min_{s\in\mathbb{R}^n} & q(s) = \langle g, s\rangle + \frac{1}{2}\langle s, Hs\rangle \\ \text{s.t.} & \|s\|_2 \leq \Delta. \end{array} \tag{1.77}$$

For simplicity, we have discarded the constant term $f(x)$ from the model as this term has no effect on the solution of the subproblem. At this point, it is important to make an observation. The solution of the subproblem lies either in the interior of the trust region ($\|s\|_2 < \Delta$), or on the boundary ($\|s\|_2 = \Delta$). If the solution is inside, the trust-region constraint is ineffective and thus the solution of the subproblem is the unconstrained minimizer of $q(s)$. But this can only happen if $q(s)$ is convex, that is if H is positive semidefinite. This simple observation suggests the idea of the algorithm. First, compute an unconstrained minimizer of $q(s)$ if it exists and is finite. If the solution is inside the trust region, accept it as the next iterate. Otherwise, the model minimizer is the minimizer of $q(s)$ on the boundary. The following theorem characterizes the solution of the subproblem and was produced independently by Gay (1981) and Sorensen (1982).

Theorem 1.4.1 (Characterization of the exact solution of the trust-region subproblem) The step s^* is a global minimizer of $q(s)$ subject to $\|s\|_2 \leq \Delta$ if and only if s^* is feasible and there is a scalar $\lambda \geq 0$ such that the following conditions are satisfied:

$$\begin{array}{l} H(\lambda)s = -g, \\ \lambda(\|s\|_2 - \Delta) = 0, \\ H(\lambda) \text{ is positive semidefinite}, \end{array} \tag{1.78}$$

where $H(\lambda) \stackrel{\text{def}}{=} H + \lambda I$. If $H(\lambda)$ is positive definite, the solution is unique.

The Moré-Sorensen method (see Hebden (1973), Gay (1981) and Moré and Sorensen (1983)) consists in iteratively solving the system $H(\lambda)s = -g$ and then applying a Newton step on parameter λ for the secular equation

$$\phi(\lambda) = \frac{1}{\|s(\lambda)\|_2} - \frac{1}{\Delta} = 0, \tag{1.79}$$

where $s(\lambda)$ is the solution of the system. Note that we prefer solving this equation rather than the simpler

$$\|s(\lambda)\|_2 - \Delta = 0 \tag{1.80}$$

since it is numerically more stable (see Conn et al. (2000) for a complete discussion of this point). Since the Newton method is not globally convergent, we add to the algorithm safeguards, that is a lower and an upper bound, λ^L and λ^U, on the parameter λ. Note that the trust-region constraint need just to be verified approximately in the convergence theory and thus, to obtain a more efficient algorithm, we just require that the equation (1.79) is solved approximately, introducing a "belt" around the trust-region radius characterized by two radii Δ^L and Δ^U with $\Delta^L < \Delta < \Delta^U$. We now present the Moré-Sorensen algorithm in Algorithm 1.4.2.

We add some comments on the algorithm:

1. By combining Rayleigh quotient inequalities, it is possible to derive good initial safeguards of parameter λ (see Section 7.3.8 of Conn et al. (2000)):

$$\lambda^L = \max\left\{0, -\min_i H_{ii}, \frac{\|g\|_2}{\Delta} - \min\{\|H\|_F, \|H\|_\infty\}\right\}, \tag{1.82}$$

and

$$\lambda^U = \max\left\{0, \frac{\|g\|_2}{\Delta} + \min\{\|H\|_F, \|H\|_\infty\}\right\}. \tag{1.83}$$

If the initial λ^L is zero, a good candidate for the initial parameter λ is also zero so that the interior solution is first checked. A good way to initialize the belt is to take

$$\begin{array}{rcl} \Delta^L & = & (1-\theta)\Delta \\ \Delta^U & = & (1+\theta)\Delta, \end{array} \tag{1.84}$$

with $\theta = 0.1$.

Algorithm 1.4.2: Moré-Sorensen algorithm $[s, \lambda]$=**MS**(g, H, Δ)

Step 0: Initialization. Compute λ^L, λ^U, λ, Δ^L and Δ^U. Set $k = 0$.

Step 1: Factorization. Attempt a Cholesky factorization of $H(\lambda)$. If it is not possible, set $\lambda^L = \lambda$, pick $\lambda \in (\lambda^L, \lambda^U]$ and set $k = k + 1$.

Step 2: System solution. Solve $LL^T s = -g$, where L is the Cholesky factor of $H(\lambda)$.

Step 3: Termination test. If either $\lambda \approx 0$ and $\|s\|_2 \leq \Delta^U$ or $\Delta^L \leq \|s\|_2 \leq \Delta^U$, exit.

Step 4: Newton step. Solve $Lw = s$ and compute

$$\lambda^+ = \lambda + \left(\frac{\|s\|_2 - \Delta}{\Delta}\right)\left(\frac{\|s\|_2^2}{\|w\|_2^2}\right). \tag{1.81}$$

Step 5: Bracket update. If $\|s\|_2 > \Delta^U$, set $\lambda^L = \lambda$. If $\|s\|_2 < \Delta^L$, set $\lambda^U = \lambda$.

Step 6: Parameter update. If $\lambda^+ \in (\lambda^L, \lambda^U)$, set $\lambda = \lambda^+$. Otherwise pick $\lambda \in (\lambda^L, \lambda^U)$. Set $k = k + 1$ and go back to Step 1.

2. If the Cholesky factorization of the Hessian is not possible, it means that the matrix $H(\lambda)$ is not positive semidefinite and thus the parameter λ is not large enough. We thus take λ as a lower bound.

3. A good way to pick the parameter λ inside the interval given by its safeguards is to take

$$\lambda = \max\left\{\sqrt{\lambda^L \lambda^U}, \lambda^L + \psi(\lambda^U - \lambda^L)\right\}, \tag{1.85}$$

   with $\psi = 0.01$, which guarantees a good decrease in the length of the interval.

4. The check for termination of the algorithm corresponds to the two possible solutions of the trust-region subproblem, that is an interior solution with a positive definite Hessian or a boundary solution.

5. The definition of λ^+ in Step 4 corresponds to applying a Newton step on the secular equation (1.79). Note that it requires the solution of a triangular linear system.

6. The update of the safeguards in Step 5 is quite simple to understand. If the step norm is bigger than the trust-region radius, it means that the matrix $H(\lambda)$ is not positive definite enough, and thus we set the current parameter λ as a lower bound. On the opposite, if the step norm is smaller than the trust-region radius, it means that the matrix $H(\lambda)$ is too much positive definite, and thus we set the current parameter λ as an upper bound.

This method is quite effective since it solves trust-region subproblems generally in 5 or 6 iterations. The main problem is that it requires at each iteration the computation of a Cholesky

factorization. Thus, this method is clearly to be used for problems of small size or for which efficient sparse Cholesky factorization is possible but is not to be applied on larger problems. In this case, we prefer using the truncated conjugate gradient algorithm presented in the next section. Note however that Dollar, Gould and Robinson (2009) have proposed recently new techniques to improve the method.

1.4.2.2 The Truncated conjugate gradient algorithm

Before presenting the truncated conjugate gradient algorithm, we briefly recall the conjugate gradient method for linear systems. The method consists in solving a linear system made up with a symmetric positive definite matrix and this method is due to Hestenes and Stiefel (1952). This process is performed without storing any additional matrix, even the matrix of the system. As a consequence one of the key properties of these methods is that they require little storage.

The basis of conjugate-gradient algorithms is the notion of conjugate directions. A set of nonzero vectors $\{d_1, \ldots, d_k\}$ is said to be *conjugate* with respect to a symmetric positive definite matrix A, if

$$d_i^T A d_j \quad \text{for all } i \text{ and } j \text{ such that } i \neq j. \tag{1.86}$$

The conjugate directions can be used to solve a quadratic minimization problem of the form

$$\min_{x \in \mathbb{R}^n} q(x) = \frac{1}{2} x^T A x + b^T x, \tag{1.87}$$

where A is supposed to be symmetric and positive definite. The unique solution to this problem is also the unique solution to the system of linear equations

$$Ax = -b. \tag{1.88}$$

The conjugate-gradient algorithm is obtained by choosing the successive direction vectors as a conjugate version of the successive residuals of the system obtained as the method progresses. The directions are determined sequentially at each step of the iteration. At iteration k one evaluates the current negative residual vector and it is added a linear combination of the previous direction vectors to obtain a new conjugate direction vector along which to move. The conjugate-gradient method is described in the Algorithm 1.4.3 and its convergence must occur by the n-th iteration (at the latest) in exact arithmetic.

Consider now the unconstrained trust-region subproblem

$$\begin{array}{ll} \min_{s \in \mathbb{R}^n} & q(s) = \langle g, s \rangle + \frac{1}{2} \langle s, Hs \rangle \\ \text{s.t.} & \|s\|_2 \leq \Delta. \end{array} \tag{1.90}$$

Suppose that we apply Algorithm 1.4.3 to the minimization of $q(s)$ regardless of whether or not H is positive definite or whether or not the generated iterates remain in the trust region. Then three possibilities might happen. Firstly, the curvature $\langle d_k, Hd_k \rangle$ may be positive at each

Algorithm 1.4.3: Conjugate-gradient method

Given a starting point x_0 , set $r_0 = Ax_0 + b$ and $d_0 = -r_0$. Until convergence, perform the following sequence of operations

$$\begin{aligned} \alpha_k &= \frac{r_k^T r_k}{d_k^T A d_k} \\ x_{k+1} &= x_k + \alpha_k d_k \\ r_{k+1} &= r_k + \alpha_k A d_k \\ \beta_k &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\ d_{k+1} &= -r_{k+1} + \beta_k d_k \\ k &= k+1 \end{aligned} \tag{1.89}$$

iteration while the iterates remain in the trust region. This corresponds to the convex interior-solution discussed in the previous section. Secondly, we can have that $\langle d_k, Hd_k\rangle \leq 0$ at iteration k. In this case, the model is not convex and the computed step does not give a reduction in q. But our aim is to minimize the model inside the trust-region and thus, we will minimize q along the current direction while staying in the trust region which can be done by computing the positive root of the quadratic equation

$$\|x_k + \alpha d_k\|^2 = \Delta^2. \tag{1.91}$$

The third possibility is that the iterate lies outside the trust region at iteration k. In this case, we consider a nice property of the conjugate gradient algorithm presented in Theorem 1.4.2 which was proved by Steihaug (1983).

Theorem 1.4.2 Suppose that the conjugate gradient algorithm is applied to minimize $q(s)$ starting from $s_0 = 0$, and $\langle d_i, Hd_i\rangle > 0$ for $0 \leq i \leq k$. Then the iterates x_j satisfy the inequalities

$$\|s_j\| < \|s_{j+1}\| \quad \text{for } 0 \leq j \leq k-1. \tag{1.92}$$

This ensures that if an iterate lies outside the trust region, the following iterates will never come back in the trust region. Since we have to stay inside the trust region, as in the second possibility, we minimize q along the current direction while staying in the trust region again solving (1.91). These ideas are summarized in Algorithm 1.4.4 where we have denoted the iterates s_k to reflect that we are seeking the step and not the next iterate. This Algorithm is due to Toint (1981), Steihaug (1983) and Dembo, Eisenstat and Steihaug (1982).

In Algorithm 1.4.4, the question of the accuracy remains open. Actually, any iterate is sufficient to ensure convergence to a first order critical point. This results from the fact that the

Algorithm 1.4.4: The Steihaug-Toint truncated conjugate gradient (TCG) algorithm(g, H, Δ)

Step 0: Initialization. Let $s_0 = 0$, $g_0 = g$, $d_0 = -g_0$ and $k = 0$.

Step 1: First computations. Compute $\kappa_k = \langle d_k, Hd_k \rangle$ and $\alpha_k = \frac{g_k^T g_k}{\kappa_k}$.

Step 2: Termination test. If accuracy, exit. Otherwise if $\kappa_k \leq 0$ of if $\|s_k + \alpha_k d_k\| \geq \Delta$, compute σ_k as the positive root of $\|s_k + \sigma d_k\|^2 = \Delta^2$, set $s_{k+1} = s_k + \sigma_k d_k$ and exit.

Step 3: Conjugate gradient. Set

$$\begin{array}{rcl} s_{k+1} & = & s_k + \alpha_k d_k, \\ g_{k+1} & = & g_k + \alpha_k H d_k, \\ \beta_k & = & \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}, \\ d_{k+1} & = & -g_{k+1} + \beta_k d_k, \\ k & = & k + 1, \end{array} \tag{1.93}$$

and go to Step 1.

first iterate is actually the Cauchy point of the model and the following iterates give lower values of the model. Although a common stopping criterion is to stop as an iteration k is reached for which

$$\|g_k\| \leq \|g_0\| \min\{\zeta, \|g_0\|^\theta\} \quad \text{or } k > k_{\max}. \tag{1.94}$$

where $\zeta < 1$, $\theta \geq 0$ and $k_{\max} \geq 0$. Note that if $\theta > 0$ and a suitable model Hessian is used , superlinear convergence of the underlying trust-region method is possible. Values like $\zeta = 0.1$, $\theta = 0.5$ and $k_{\max} = n$ are typical. Note also that in the strictly convex case, Yuan (2000) has proved that the decrease obtained with Algorithm 1.4.4 is at least half the decrease obtained by minimizing the model exactly in the trust region.

We now consider the bound-constrained trust-region subproblem

$$\begin{array}{ll} \min_{s \in \mathbb{R}^n} & q(s) = \langle g, s \rangle + \frac{1}{2} \langle s, Hs \rangle \\ \text{s.t.} & \|s\|_\infty \leq \Delta \\ & l \leq x + s \leq u. \end{array} \tag{1.95}$$

Note that we use an infinity norm to define the trust region rather than an Euclidean one since it is more easy to combine with the bound constraints. Indeed, if we define

$$l^\Delta \stackrel{\text{def}}{=} \max(-\Delta, l - x), \tag{1.96}$$

and

$$u^\Delta \stackrel{\text{def}}{=} \min(\Delta, u - x), \tag{1.97}$$

we could rewrite the subproblem only with bound constraints on the step size

$$\begin{array}{ll} \min_{s\in\mathbb{R}^n} & q(s) = \langle g, s\rangle + \frac{1}{2}\langle s, Hs\rangle \\ \text{s.t.} & l^\Delta \leq s \leq u^\Delta. \end{array} \tag{1.98}$$

Before introducing the algorithm, note that a variable is fixed at its constraint value if its index belongs to the binding set $\mathcal{B}(s)$ of the problem (see Equation (1.47)). Let us denote the free vector of vector z by $P_{\mathcal{B}^c}(z)$ where

$$P_{\mathcal{B}^c}(z)_i \stackrel{\text{def}}{=} \begin{cases} z_i & \text{if the } i \notin \mathcal{B}(s), \\ 0 & \text{otherwise.} \end{cases} \tag{1.99}$$

Solving the subproblem corresponds to minimizing the quadratic $q(s)$ inside an hypercube defined by the bounds. The projected truncated conjugate gradient algorithm consists in minimizing the quadratic $q(s)$ by a conjugate gradient algorithm while remaining on a face of the hypercube. If the algorithm leaves a face, it is relaunched on the new face. The projected truncated conjugate gradient algorithm is presented in Algorithm 1.4.5 where we denote $z_{k,i}$ the i-th component of vector z_k.

We now make some comments on Algorithm 1.4.5:

1. The algorithm consists in three loops

   (a) The first loop on the faces between Step 1 and Step 6. In this loop, we have identified a number of free variables and we explore the "restricted" problem in order to drive the restricted gradient to zero. If, inside the loop, we identify new fixed variables (this is checked in Step 5 of the algorithm), we restart the algorithm on the new face of the hypercube with the steepest descent direction. Note that in this case, the number of conjugate gradient iterations is not reset such that we have a control on the maximum number of conjugate gradient iterations. We exit the algorithm if either a maximum number of faces has been explored or if the norm of the free gradient is small enough.

   (b) The second loop is the classical conjugate gradient loop between Step 2 and Step 6. In this loop, we perform the conjugate gradient until termination is observed or until we reach a new face of the hypercube defined by the bound constraints. We exit the algorithm if the maximum number of conjugate gradient iterations has been reached, if the step length is too small or if the norm of the free gradient is small enough.

   (c) Finally, the third loop is the loop to compute the step length between Step 3 and Step 4. In this loop, the step length is computed as the minimum between the length obtained by minimizing the quadratic without constraints and the length to reach the bound for each variable. If a bound is first reached, then we add its index to the binding set and we restart the computation of the step length with this variable fixed at its constraint. Otherwise, if the minimum of the quadratic is first reached

(which is identified by the test $i_{\min} = 0$ in Step 4), then we exit this loop. Note that we could also exit this loop if there is no free variable left, or if the direction is no more a descent direction, or if a maximum number of iterations is reached or if the step length is not large enough. Note also that the algorithm allows the case where multiple bounds are reached at a same iteration by recomputing the binding set at the end of Step 4.

2. To compute the accuracy threshold ϵ_g, we could use the right hand side of the inequality (1.94). The tolerance on the step size is a small constant (10^{-12}). The maximum number of faces to explore is usually 2 or 3 and the maximum number of iterations $k_{\max}$ and $j_{\max}$ is usually n.

Now that we know how to solve the trust-region subproblem either for small or large scale problems, the basic trust-region algorithm is more clear and it is possible to implement it quite efficiently. However, for more than thirty years, people have developed techniques to make this algorithm even better. We present some of these refinements in the next section.

Algorithm 1.4.5: The prolongated truncated conjugate gradient (PTCG) algorithm$(g, H, l^{\Delta}, u^{\Delta})$

Step 0: Initialization. Set $face = 0$, $s_0 = 0$, $g_0 = g$ and $k = 0$. Compute ϵ_g and ϵ_α.

Step 1: Face loop. Compute $\mathcal{B}(s_k)$ and $F = \sharp\mathcal{B}(s_k)$. Set $d_k = -P_{\mathcal{B}^c}(g_k)$ and $n_g = \|P_{\mathcal{B}^c}(g_k)\|$. If $face \geq face_{\max}$ or $n_g \leq \epsilon_g$, exit.

Step 2: CG loop. Set $j = 0$, $\sigma_j = s_k$, $\delta_j = d_k$ and $\gamma_j = g_k$.

Step 3: Step length. If $\sharp\mathcal{B}(\sigma_j) = 0$ or $\gamma_j^T \delta_j \geq 0$ or $j > j_{\max}$ or $\alpha < \epsilon_\alpha$, go to Step 5. Else, compute $\kappa_j = \delta_j^T H \delta_j$,

$$\alpha_0 \stackrel{\text{def}}{=} \begin{cases} \frac{-\gamma_j^T \delta_j}{\kappa_j} & \text{if } \kappa_j > 0, \\ +\infty & \text{otherwise}, \end{cases} \tag{1.100}$$

and

$$\alpha_i \stackrel{\text{def}}{=} \begin{cases} \frac{u_i^{\Delta} - \sigma_{j,i}}{\delta_{j,i}} & \text{if } \delta_{j,i} > 0 \\ \frac{l_i^{\Delta} - \sigma_{j,i}}{\delta_{j,i}} & \text{if } \delta_{j,i} < 0 \\ +\infty & \text{otherwise} \end{cases} \qquad 1 \leq i \leq n. \tag{1.101}$$

Set $\alpha = \min_{0 \leq i \leq n} \alpha_i$ and $i_{\min}$ its index.

Step 4: Variables update. Set

$$\begin{array}{rcl} \sigma_{j+1} & = & \sigma_j + \alpha \delta_j, \\ \gamma_{j+1} & = & \gamma_j + \alpha H \delta_j. \end{array} \tag{1.102}$$

If $i_{\min} = 0$, go to Step 5. Otherwise compute $\mathcal{B}(\sigma_{j+1})$, set $\delta_{j+1} = P_{\mathcal{B}^c}(\delta_j)$, $j = j + 1$ and go to Step 3.

Step 5: CG termination. If $\sharp\mathcal{B}(\sigma_{j+1}) \neq F$, set $face = face + 1$, $s_{k+1} = \sigma_{j+1}$, $g_{k+1} = \gamma_{j+1}$, $k = k + 1$ and go to Step 1. Otherwise, if $\|P_{\mathcal{B}^c}(\gamma_{j+1})\| \leq \epsilon_g$ or $\alpha < \epsilon_\alpha$ or $k \geq k_{\max}$, exit.

Step 6: CG direction update. Compute

$$\begin{array}{rcl} s_{k+1} & = & \sigma_{j+1}, \\ g_{k+1} & = & \gamma_{j+1}, \\ \beta_{k+1} & = & \left(\frac{\|P_{\mathcal{B}^c}(g_{k+1})\|}{n_g} \right)^2, \\ d_{k+1} & = & -P_{\mathcal{B}^c}(g_{k+1}) + \beta_{k+1} \delta_{j+1}, \\ n_g & = & \|P_{\mathcal{B}^c}(g_{k+1})\|, \\ k & = & k + 1, \end{array} \tag{1.103}$$

and go to Step 2.

1.5 Refinements of the Basic Trust-Region algorithm

We now present possible refinements of the basic trust-region algorithm to obtain the most possible efficient algorithm. Indeed, the algorithm exists for more than thirty years and mathematicians have developed their own versions of the algorithm. The consequence is that there exist many different algorithms called "basic trust-region" algorithm. In this thesis, we have thus make numerical tests in order to test in the same framework all these variants.

1.5.1 Numerical considerations

We assume that we use the second order Taylor model defined in (1.69) and that we use an Euclidean norm to define the trust region (note that in this thesis we always use the Euclidean norm if we consider an unconstrained problem and the infinity norm if we consider a bound-constrained problem).

We stop our algorithm with success at iteration k if the gradient norm is smaller than a given threshold or if the function value is beyond a value, that is, if

$$\|g_k\|_2 < \epsilon_g \quad \text{or} \quad f_k < f_{\min}, \tag{1.104}$$

where $f_k = f(x_k)$, ϵ_g is the accuracy threshold, $f_{\min}$ is the value under which the objective function is supposed unbounded below. We stop our algorithm with failure at iteration k if the iteration limit has been reached or if the step length is too small, that is, if

$$k \geq k_{\max} \quad \text{or} \quad \|s_k\|_2 < \epsilon_s, \tag{1.105}$$

where $k_{\max}$ is the limit on the number of iterations and ϵ_s is the value under which the step length is supposed too small. Common values used in practice for these thresholds are

$$\epsilon_g = 10^{-5}, \quad f_{\min} = -10^{20}, \quad k_{\max} = 50\,000 \quad \text{and} \quad \epsilon_s = 10^{-15}\|x_k\|_2. \tag{1.106}$$

The trust-region parameters η_1 and η_2 have already been studied extensively and we here consider the values advised in Conn et al. (2000) of

$$\eta_1 = 0.01 \quad \text{and} \quad \eta_2 = 0.95. \tag{1.107}$$

A common mistake when using a quadratic model is to compute the difference $m_k(x_k + s_k) - m_k(x_k)$ by computing both terms and then subtracting them. This is particularly unwise when both values are very large but their difference is small, since numerical rounding can cause significant cancellation errors. It is usually far better to recognize that

$$m_k(x_k + s_k) - m_k(x_k) = \langle g_k, s_k \rangle + \tfrac{1}{2}\langle s_k, H_k s_k \rangle \tag{1.108}$$

and to compute the difference by the right-hand-side of (1.108). Note that this value may be computed for nearly no cost in the truncated conjugate gradient algorithm.

One of the most dangerous stages for a trust-region method is surprisingly when it is on the point of convergence. In this case, both the numerator and denominator in the definition of ρ_k

are small and suffer from the effects of floating-point arithmetic. As an extreme example, when the differences are both at the level of machine precision, rather than having $\rho_k = 1$, we might easily see a computed $\rho_k = -1$ causing our algorithm to reduce the trust-region radius. In practice, if both differences are smaller than a small multiple of the relative machine precision, we might replace ρ_k by 1. Thus the computation of ρ_k may be obtained by Algorithm 1.5.1 where ϵ_M denotes the machine precision.

Algorithm 1.5.1: Computation of ρ_k

Step 1: Initialization. Set $\epsilon = 50\epsilon_M$ and compute $\delta_k = \epsilon \max\{1, |f_k|\}$.

Step 2: Compute the decreases. Compute

$$\begin{array}{rcl} \delta f_k & = & f(x_k + s_k) - f_k - \delta_k, \\ \delta m_k & = & \langle g_k, s_k \rangle + \frac{1}{2}\langle s_k, H_k s_k \rangle - \delta_k. \end{array} \tag{1.109}$$

Step 3: Compute the ratio. Compute

$$\rho_k = \begin{cases} 1 & \text{if } |\delta f_k| < \epsilon \text{ and } |\delta m_k| < \epsilon \text{ or if } m_k(x_k + s_k) = f(x_k + s_k) \\ \frac{\delta f_k}{\delta m_k} & \text{otherwise.} \end{cases} \tag{1.110}$$

All the tests presented in this section were performed in Matlab v. 7.1.0.183 (R14) Service Pack 3 on a 3.2 Ghz Intel single-core processor computer with 2 GB of RAM. The algorithms are every time tested on all of the 146 unconstrained problems of the CUTEr collection (see Gould et al., 2003*a*). For the problems whose dimension may be changed, we chose a reasonably small value in order not to overload the CUTEr interface with Matlab. The starting points are the standard ones provided by the CUTEr library. The minimizer of the model inside the trust-region was computed exactly using the Moré-Sorensen algorithm (and also approximately using the Steihaug-Toint algorithm in the case of the retrospective algorithm).

1.5.2 Performance profiles

In this section, we compare some versions of the algorithm. But of course we need a way to compare them. In the past, mathematicians used tables of results of their algorithms on the tested problems. However, this strategy may be difficult to use if the number of problems is large. Dolan and Moré (2002) have introduced the notion of performance profiles that we use in this thesis to compare the numerical results.

Performance profiles give, for every $\sigma \geq 1$, the proportion $p(\sigma)$ of test problems on which each considered algorithmic variant has a performance within a factor σ of the best. The performance of the algorithm may be compared for different performance measures, like the CPU-

time needed to reach the solution, the number of iterations, number of function evaluations, ...

Let us formulate this idea mathematically. Suppose that we want to compare a set of solvers $\mathcal{S}$, such as different solvers or several algorithmic options for one solver, on a set of test problems $\mathcal{P}$. Let $q_{p,s}$ denote the quantities we want to compare for problem p and run s. For these latter statistics, the smaller the value, the better the considered variant. The performance ratio is defined as

$$r_{p,s} \stackrel{\text{def}}{=} \frac{q_{p,s}}{\min\{q_{p,s} \,|\, s \in \mathcal{S}\}}, \tag{1.111}$$

in order to compare the performance of solver s on problem p with the best performance by any solver in $\mathcal{S}$ on p. Note that if the run s has failed on problem p, we set the ratio $r_{p,s}$ to infinity. Then, we define the performance profile for each solver s as

$$p_s(\sigma) \stackrel{\text{def}}{=} \frac{\sharp\{p \in \mathcal{P} \,|\, r_{p,s} \leq \sigma\}}{\sharp\{\mathcal{P}\}}, \quad \sigma \geq 1. \tag{1.112}$$

So performance profiles give, for every $\sigma \geq 1$, the proportion $p(\sigma)$ of test problems on which each considered algorithm has a performance within a factor σ of the best. For example, the value $p_s(1)$ is the probability that the solver or algorithmic option s is the best. $p_s(2)$ gives the percentage of test problems for which variant s is within a factor of 2 of the best. And the limit

$$\lim_{\sigma \to \infty} p_s(\sigma) \tag{1.113}$$

gives the fraction of test problems of $\mathcal{P}$ for which the variant s succeeded. As a consequence, the values on the left of the plot represent the efficiency of each solver and the values on the right give information about the robustness of the solvers. So visually speaking, the "best" solver is the highest on the plot.

1.5.3 Initialization of the trust-region radius

The choice of the initial trust-region radius Δ_0 is a significant issue in the algorithm. Indeed, a bad choice may lead to a waste of iterations to either increase or decrease it to an adequate value. For example, in the case of a convex quadratic function, an infinite trust-region radius is adequate. But if it is chosen too small compared with the distance between the starting point and the solution, a lot of wasteful iterations may be required, in which the model is minimized inside increasingly large, but not large enough trust regions.

But this strategy has not to be too costly. Indeed, if the considered optimization problem is well scaled, a costly strategy is a waste of time for no benefit. This is why some simple strategies have been suggested in the past such as,

$$\Delta_0 = 1 \quad \text{or} \quad \Delta_0 = \tfrac{1}{10}\|g_0\|. \tag{1.114}$$

These strategies are quite efficient if the problem considered is well scaled and moreover have the advantage to require no additional information.

Another more elaborate strategy, which takes the Hessian into account, is to choose the radius as the length of the quadratic model minimizer in the steepest descent direction when it exists, that is,

$$\Delta_0 = \frac{\|g_0\|^2}{|\langle g_0, H_0 g_0 \rangle|}, \tag{1.115}$$

where the absolute value stands for the case where the steepest descent is a direction of negative curvature.

Finally, Sartenaer (1997) has developed the most elaborate strategy to initialize the trust-region radius whose idea is to determine the largest possible radius such that the model prediction along the steepest descent direction is sufficiently close to the true objective function value, while at the same time exploiting any improvement obtained in the objective function itself during the process. This strategy requires additional evaluations of the objective function and some linear algebra. It is beyond the scope of this thesis to present in details the method but in order to have the best possible trust-region algorithm, we have implemented it and compared it to the other strategies presented in (1.114) and (1.115). The number of function evaluations is compared in Figure 1.3 by a performance profile. As we can see on this figure, the initialization of the trust-region radius has no effect on the robustness of the algorithms. But if we are looking for their efficiency, we can easily see that it is preferable to choose the initializations (1.114).

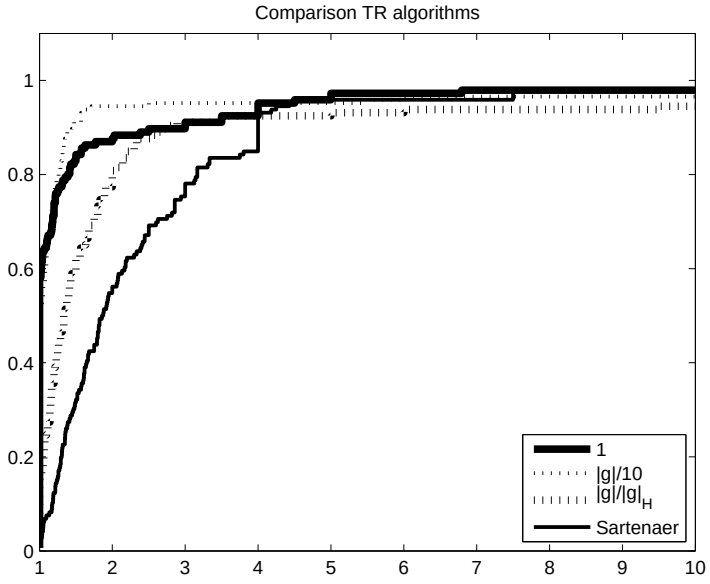

Figure 1.3: Performance profile comparing the number of function evaluations of the trust-region algorithms using different initializations of the trust-region radius.

The poor results of Sartenaer's update may appear amazing but it is easily understandable with the Figure 1.4 which compares the number of iterations of the trust-region algorithm using the same initializations. We see on this figure that this strategy is the best to use if we just look the number of iterations but the additional number of function evaluations it requires to initialize the trust-region is a waste of time for some problems and one must therefore prefer a simpler update.

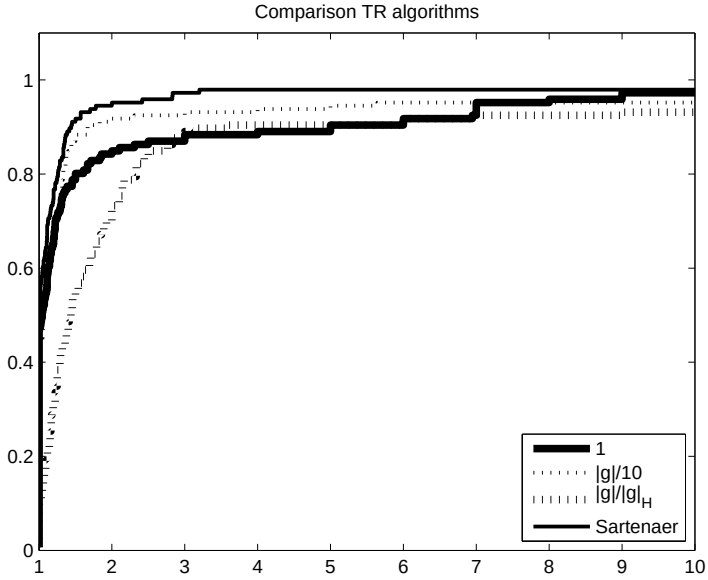

Figure 1.4: Performance profile comparing the number of iterations of the trust-region algorithms using different initializations of the trust-region radius.

1.5.4 Update of the trust-region radius

The update of the trust-region radius is also of great importance since it determines the maximum length of the next step. The framework (1.74) described in the algorithm is quite large. A quite simple strategy commonly used is

$$\Delta_{k+1} = \begin{cases} \psi_1 \Delta_k & \text{if } \rho_k \geq \eta_2, \\ \Delta_k & \text{if } \rho_k \in [\eta_1, \eta_2), \\ \psi_2 \Delta_k & \text{if } \rho_k < \eta_1, \end{cases} \tag{1.116}$$

where $\psi_1 = 2$ and $\psi_2 = 0.25$. But even if it is theoretically correct, this do not correspond to a numerically clever choice. Indeed, suppose that the iteration is unsuccessful with a step which length is smaller than the quarter of the trust-region radius. This strategy results in a new radius which is still larger than the current step and thus, the next step is identical which is a waste of work. Moreover, suppose that a lot of successful iterations occur, then the trust-region radius become quite large even if all the steps have been small. This results that if an iteration is unsuccessful, then a lot of iterations are required to decrease the radius. This is why we prefer using the following strategy to update the trust-region radius which use the step length

$$\Delta_{k+1} = \begin{cases} \max\{\alpha_1 \|s_k\|, \Delta_k\} & \text{if } \rho_k \geq \eta_2, \\ \Delta_k & \text{if } \rho_k \in [\eta_1, \eta_2), \\ \alpha_2 \|s_k\| & \text{if } \rho_k < \eta_1, \end{cases} \tag{1.117}$$

where we use the values $\alpha_1 = 2.5$ and $\alpha_2 = 0.25$ advised in Conn et al. (2000).

But further refinements are possible. If the adequation between the model and the function is so poor that $\rho_k < 0$, we might consider how large the radius would need to be to allow us to make a very successful step if the step taken is in the direction we have just attempted and if the true function is a quadratic. By a simple interpolation calculation, we should choose

$$\Delta_{k+1} = \theta_k \Delta_k \tag{1.118}$$

with

$$\theta_k \stackrel{\text{def}}{=} \frac{(1-\eta_2)\langle g_k, s_k\rangle}{(1-\eta_2)[f_k + \langle g_k, s_k\rangle] + \eta_2 m_k(x_k + s_k) - f(x_k + s_k)}. \tag{1.119}$$

We could safeguard this formula by introducing the parameter $\gamma_1 = 0.0625$. This gives us the following radius update

$$\Delta_{k+1} = \begin{cases} \max\{\alpha_1 \|s_k\|, \Delta_k\} & \text{if } \rho_k \geq \eta_2, \\ \Delta_k & \text{if } \rho_k \in [\eta_1, \eta_2), \\ \alpha_2 \|s_k\| & \text{if } \rho_k \in [0, \eta_1), \\ \min\{\alpha_2 \|s_k\|, \max\{\gamma_1, \theta_k\}\Delta_k\} & \text{if } \rho_k < 0. \end{cases} \tag{1.120}$$

One can also try to use this formula to extrapolate the value of Δ_{k+1} in the case of a very successful iteration but this strategy does not appear computationally effective in our experiments. We thus compare the step-based update without interpolation and the step-based update with interpolation in a performance profile in Figure 1.5 where the number of function evaluations is the measure of performance (the simple update presented in 1.116 appears not effective in our experiments and we have chosen to not represent these results to not overload the performance profile).

1.5.5 A retrospective algorithm

1.5.5.1 The method

The retrospective method is motivated by applications in adaptive techniques which exploit the information made available during the optimization process in order to vary the accuracy of the objective function computation. These techniques typically appear in the context of a noisy objective function, where noise reduction can be achieved but at a significant cost. A first trust-region method with dynamic accuracy is described in Section 10.6 of Conn et al. (2000). The main idea there is to impose a model reduction larger than some multiple of the noise evaluated at both the current and candidate iterates. A cheaper nonmonotone approach has been developed in the context of nonlinear stochastic programming by Bastin, Cirillo and Toint (2006*b*), (see also Bastin, Cirillo and Toint, 2006*a*) more specifically for the minimization of sample average approximations (Shapiro, 2003) relying on Monte-Carlo sampling, a method also known as sample-path optimization (Robinson, 1996). The main difference with respect to the work of Conn *et al.* is that it allows a reduction of the model smaller than the noise level. In both cases, the size of the model reduction is the main component to decide on the desired accuracy of the objective function: the adaptive mechanism is thus applied on the basis of past

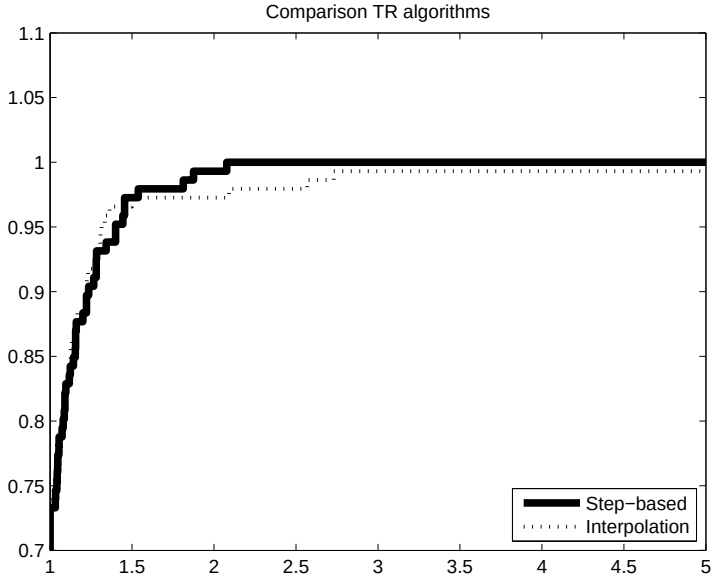

Figure 1.5: Performance profile comparing the number of function evaluations of the trust-region algorithms using different updates of the trust-region radius.

information, at the previous iterate, rather that at the current one. Our new proposal is then motivated by the hope of improving these techniques because the most relevant information on the model's quality at the current iterate would be used, instead of at the previous iterate.

The retrospective algorithm differs in that the trust-region radius is updated after each successful iteration k (that is at the beginning of iteration $k+1$) on the basis of the *retrospective* ratio

$$\tilde{\rho}_{k+1} \stackrel{\text{def}}{=} \frac{f(x_{k+1}) - f(x_{k+1} - s_k)}{m_{k+1}(x_{k+1}) - m_{k+1}(x_{k+1} - s_k)} = \frac{f(x_k) - f(x_k + s_k)}{m_{k+1}(x_k) - m_{k+1}(x_k + s_k)}$$

of achieved to predicted changes, while continuing to use ρ_k to decide whether the trial iterate may be accepted. Our method therefore distinguishes the two roles played by ρ_k in the classical algorithm: that of deciding acceptance of the trial iterate and that of determining the radius update. It also explicitly takes into account that m_{k+1}, not m_k, is used within the trust region of radius Δ_{k+1}. Thus, when the iterate has first been accepted, that is when $\rho_k \geq \eta_1$, we compute this radius by either increasing the current radius or leaving it unchanged if $\tilde{\rho}_k \geq \tilde{\eta}_1$ or decrease it otherwise. In this last case, it is again chosen in the interval $[\gamma_0 \|s_k\|, \gamma_1 \|s_k\|]$. Moreover, when $\tilde{\rho}_k$ is negative, a quadratic fit of the model is used to determine a tentative new radius which will make the next iteration very successful in the sense that $\tilde{\rho}_{k+1} \geq \tilde{\eta}_2$ for some $\tilde{\eta}_2 \in (\tilde{\eta}_1, 1)$. This value is given by $\tilde{\theta}_{k+1}\Delta_k$, where

$$\tilde{\theta}_{k+1} \stackrel{\text{def}}{=} \frac{-(1-\tilde{\eta}_2)\langle \nabla_x f(x_{k+1}), s_k \rangle}{(1-\tilde{\eta}_2)[f(x_{k+1}) - \langle \nabla_x f(x_{k+1}), s_k \rangle] + \tilde{\eta}_2 m_{k+1}(x_k) - f(x_k)}. \tag{1.121}$$

Notice that $\tilde{\theta}_{k+1}$ uses the gradient at the new point, rather than the old one as in (1.119).

This leads to the retrospective trust-region method described as Algorithm 1.5.2. The precise definitions of the model (at Step 1) and of "sufficient reduction" (at Step 3) are described

more in details in the convergence theory that is presented in the next section.

Algorithm 1.5.2: Retrospective trust-region algorithm (RTR)

Step 0: Initialization. An initial point x_0 and initial trust-region radius $\Delta_0 > 0$ are given. The constants η_1, $\tilde{\eta}_1$, $\tilde{\eta}_2$, γ_0, γ_1 and γ_2 are also given and satisfy $0 < \eta_1 < 1$, $0 < \tilde{\eta}_1 \leq \tilde{\eta}_2 < 1$ and $0 < \gamma_0 < \gamma_1 \leq 1 \leq \gamma_2$. Compute $f(x_0)$ and set $k = 0$.

Step 1: Model definition. Select a twice-continuously differentiable model m_k defined in $\mathcal{B}_k$.

Step 2: Retrospective trust-region radius update. If $k = 0$, go to Step 3. If $x_k = x_{k-1}$, then choose

$$\Delta_k = \begin{cases} \gamma_1 \|s_{k-1}\| & \text{if } \rho_{k-1} \in [0, \eta_1), \\ \min[\gamma_1 \|s_{k-1}\|, \max[\gamma_0, \theta_{k-1}]\Delta_{k-1}] & \text{if } \rho_{k-1} < 0, \end{cases} \tag{1.122}$$

where θ_{k-1} is defined in (1.119). Else, define

$$\tilde{\rho}_k = \frac{f(x_{k-1}) - f(x_k)}{m_k(x_{k-1}) - m_k(x_k)} \tag{1.123}$$

and choose

$$\Delta_k = \begin{cases} \max[\gamma_2 \|s_{k-1}\|, \Delta_{k-1}] & \text{if } \tilde{\rho}_k \geq \tilde{\eta}_2, \\ \Delta_{k-1} & \text{if } \tilde{\rho}_k \in [\tilde{\eta}_1, \tilde{\eta}_2), \\ \gamma_1 \|s_{k-1}\| & \text{if } \tilde{\rho}_k \in [0, \tilde{\eta}_1), \\ \min[\gamma_1 \|s_{k-1}\|, \max[\gamma_0, \tilde{\theta}_k]\Delta_{k-1}] & \text{if } \tilde{\rho}_k < 0, \end{cases} \tag{1.124}$$

where $\tilde{\theta}_k$ is defined in (1.121).

Step 3: Step calculation. Compute a step s_k that "sufficiently reduces the model" m_k and such that $x_k + s_k \in \mathcal{B}_k$.

Step 4: Acceptance of the trial point. Compute $f(x_k + s_k)$ and define

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}. \tag{1.125}$$

If $\rho_k \geq \eta_1$, then define $x_{k+1} = x_k + s_k$ and compute $\nabla_x f(x_{k+1})$; otherwise define $x_{k+1} = x_k$. Increment k by 1 and go to Step 1.

1.5.5.2 Convergence theory

We now investigate the convergence properties of our algorithm. Since it can be considered as a variant of the basic trust-region method of Conn et al. (2000), we expect similar results and significant similarities in their proofs. In what follows, we have attempted to be explicit on the assumptions and properties, but to refer to Chapter 6 of this reference whenever possible.

Our assumptions are identical to those used for the basic trust-region method.

A.1 The Hessian of the objective function $\nabla_{xx} f$ is uniformly bounded, i.e. there exists a positive constant κ_{ufh} such that, for all $x \in \mathbb{R}^n$,

$$\|\nabla_{xx} f(x)\| \leq \kappa_{\text{ufh}}.$$

A.2 The model m_k is first-order coherent with the function f at each iteration x_k, i.e. their values and gradients are equal at x_k for all k:

$$m_k(x_k) = f(x_k) \quad \text{and} \quad g_k \stackrel{\text{def}}{=} \nabla_x m_k(x_k) = \nabla_x f(x_k).$$

A.3 The Hessian of the model $\nabla_{xx} m_k$ is uniformly bounded, i.e. there exists a constant $\kappa_{\text{umh}} \geq 1$ such that, for all $x \in \mathbb{R}^n$ and for all k,

$$\|\nabla_{xx} m_k(x)\| \leq \kappa_{\text{umh}} - 1.$$

A.4 The decrease on the model m_k is at least as much as a fraction of that obtained at the Cauchy point; i.e. there exists a constant $\kappa_{\text{mdc}} \in (0, 1)$ such that, for all k,

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{\text{mdc}} \|g_k\| \min\left[\frac{\|g_k\|}{\beta_k}, \Delta_k\right]$$

with $\beta_k \stackrel{\text{def}}{=} 1 + \max_{x \in \mathcal{B}_k} \|\nabla_{xx} m_k(x)\|$.

Note that A.4 specifies the notion of "sufficient reduction" used in Step 3 of our algorithm, while the choice of m_k in Step 1 is limited by A.2 and A.3. We also note that $s_k \neq 0$ whenever $g_k \neq 0$ because of A.4.

1.5.5.2.1 Convergence to First-Order Critical Points In this section, we prove that the retrospective trust-region algorithm is globally convergent to first-order critical points, in the sense that every limit point x_* of the sequence of iterates (x_k) produced by the algorithm 1.5.2 satisfies

$$\nabla_x f(x_*) = 0$$

irrespective of the choice of the starting point x_0 and initial trust-region radius Δ_0.

We first give a bound on the error between the true objective function f and its current model m_k at the previous iterate x_{k-1}.

Theorem 1.5.1 Suppose that A.1–A.3 hold. Then,

$$|f(x_k) - m_{k-1}(x_k)| \leq \kappa_{\rm ubh} \Delta_{k-1}^2 \tag{1.126}$$

and, if iteration $k-1$ is successful,

$$|f(x_{k-1}) - m_k(x_{k-1})| \leq \kappa_{\rm ubh} \Delta_{k-1}^2 \tag{1.127}$$

where

$$\kappa_{\rm ubh} \stackrel{\rm def}{=} \max[\kappa_{\rm ufh}, \kappa_{\rm umh}]. \tag{1.128}$$

Proof. The bound (1.126) directly results from Theorem 6.4.1 in Conn et al. (2000). We thus only prove (1.127). Because the objective function and the model are C^2 functions, we may apply the mean value theorem on the objective function f and on the model m_k, and obtain from $x_{k-1} = x_k - s_{k-1}$ that

$$\begin{aligned} f(x_{k-1}) &= f(x_k) - \langle s_{k-1}, \nabla_x f(x_k) \rangle + \tfrac{1}{2} \langle s_{k-1}, \nabla_{xx} f(\xi_k) s_{k-1} \rangle & (1.129) \\ m_k(x_{k-1}) &= m_k(x_k) - \langle s_{k-1}, g_k \rangle + \tfrac{1}{2} \langle s_{k-1}, \nabla_{xx} m_k(\zeta_k) s_{k-1} \rangle & (1.130) \end{aligned}$$

for some ξ_k, ζ_k in the segment $[x_{k-1}, x_k]$.

Because of A.2, the objective function f and the model m_k have the same value and gradient at x_k. Thus, subtracting (1.130) from (1.129) and taking absolute values yields that

$$\begin{aligned} |f(x_{k-1}) - m_k(x_{k-1})| &= \tfrac{1}{2} |\langle s_{k-1}, \nabla_{xx} f(\xi_k) s_{k-1} \rangle - \langle s_{k-1}, \nabla_{xx} m_k(\zeta_k) s_{k-1} \rangle| \\ &\leq \tfrac{1}{2} [|\langle s_{k-1}, \nabla_{xx} f(\xi_k) s_{k-1} \rangle| + |\langle s_{k-1}, \nabla_{xx} m_k(\zeta_k) s_{k-1} \rangle|] \\ &\leq \tfrac{1}{2} (\kappa_{\rm ufh} + \kappa_{\rm umh} - 1) \|s_{k-1}\|^2 \\ &\leq \tfrac{1}{2} (\kappa_{\rm ufh} + \kappa_{\rm umh} - 1) \Delta_{k-1}^2, & (1.131) \end{aligned}$$

where we successively used the triangle inequality, the Cauchy-Schwarz inequality, the induced matrix norm properties, A.1, A.3, and the fact that $x_k \in \mathcal{B}_{k-1}$ implies that $\|s_{k-1}\| \leq \Delta_{k-1}$. So (1.127) clearly holds. □

Thus the analog of Theorem 6.4.1 of Conn et al. (2000) holds in our case, where we replace the forward difference $f(x_{k+1}) - m_k(x_{k+1})$ by its retrospective variant $f(x_{k-1}) - m_k(x_{k-1})$.

As our new ratio $\tilde{\rho}_k$ uses the reduction in m_k instead of the reduction in m_{k-1}, we are interested in a bound on their difference, which is provided by this next result.

Lemma 1.5.2 Suppose that A.1–A.3 hold. Then, for every successful iteration $k-1$,

$$|\,[m_{k-1}(x_{k-1}) - m_{k-1}(x_k)] - [m_k(x_{k-1}) - m_k(x_k)]\,| \leq 2\kappa_{\rm ubh} \Delta_{k-1}^2. \tag{1.132}$$

Proof. Using the model differentiability, we apply the mean value theorem on the model m_{k-1}, and we obtain that

$$m_{k-1}(x_k) = m_{k-1}(x_{k-1}) + \langle s_{k-1}, g_{k-1}\rangle + \tfrac{1}{2}\langle s_{k-1}, \nabla_{xx} m_{k-1}(\psi_{k-1}) s_{k-1}\rangle \quad (1.133)$$

for some ψ_{k-1} in the segment $[x_{k-1}, x_k]$. Remember that (1.130) in the previous proof gives that

$$m_k(x_{k-1}) = m_k(x_k) - \langle s_{k-1}, g_k\rangle + \tfrac{1}{2}\langle s_{k-1}, \nabla_{xx} m_k(\zeta_k) s_{k-1}\rangle \quad (1.134)$$

for some ζ_k in the segment $[x_{k-1}, x_k]$. Substituting (1.133) and (1.134) inside the left-hand side of (1.132), and using A.3, the triangle inequality, the Cauchy-Schwarz inequality, and the induced matrix norm properties yield that

$$\begin{aligned} &|\,[m_{k-1}(x_{k-1}) - m_{k-1}(x_k)] - [m_k(x_{k-1}) - m_k(x_k)]\,| \\ &\quad = \left|-\langle s_{k-1}, g_{k-1} - g_k\rangle - \tfrac{1}{2}\left(\langle s_{k-1}, \nabla_{xx} m_{k-1}(\psi_{k-1}) s_{k-1}\rangle + \langle s_{k-1}, \nabla_{xx} m_k(\zeta_k) s_{k-1}\rangle\right)\right| \\ &\quad \leq \|s_{k-1}\| \cdot \|g_{k-1} - g_k\| + \kappa_{\text{umh}} \|s_{k-1}\|^2. \end{aligned} \quad (1.135)$$

Now observe that, because of A.2, $\|g_{k-1} - g_k\| = \|\nabla_x f(x_{k-1}) - \nabla_x f(x_k)\|$. We then apply the mean value theorem on $\nabla_x f$ and obtain that

$$\nabla_x f(x_k) = \nabla_x f(x_{k-1}) + \int_0^1 \nabla_{xx} f(x_{k-1} + \alpha s_{k-1}) s_{k-1}\, d\alpha. \quad (1.136)$$

Thus the Cauchy-Schwarz inequality, and A.1 give that

$$\|g_{k-1} - g_k\| \leq \int_0^1 \|\nabla_{xx} f(x_{k-1} + \alpha s_{k-1})\| \cdot \|s_{k-1}\|\, d\alpha \leq \int_0^1 \kappa_{\text{ufh}} \|s_{k-1}\|\, d\alpha = \kappa_{\text{ufh}} \|s_{k-1}\|. \quad (1.137)$$

Substituting this bound in (1.135), we obtain that

$$|\,[m_{k-1}(x_{k-1}) - m_{k-1}(x_k)] - [m_k(x_{k-1}) - m_k(x_k)]\,| \leq (\kappa_{\text{ufh}} + \kappa_{\text{umh}}) \|s_{k-1}\|^2 = 2\kappa_{\text{ubh}} \Delta_{k-1}^2$$

where we finally use (1.128), and the fact that $x_k \in \mathcal{B}_{k-1}$. □

We conclude from this result that the denominators in the expression of $\tilde{\rho}_k$ and ρ_{k-1} differ by a quantity which is of the same order as the error between the model and the objective function. Using this observation, we are now capable of showing that the iteration must be successful if the radius is sufficiently small compared to the gradient, and also that the trust-region radius has to increase in this case.

Theorem 1.5.3 Suppose that A.1–A.4 hold. Suppose furthermore that $g_k \neq 0$ and that

$$\Delta_{k-1} \leq \min\left[1 - \eta_1, \frac{(1 - \tilde{\eta}_2)}{(3 - 2\tilde{\eta}_2)}\right] \frac{\kappa_{\text{mdc}}}{\kappa_{\text{ubh}}} \|g_{k-1}\|. \quad (1.138)$$

Then iteration $k - 1$ is successful and

$$\Delta_k \geq \Delta_{k-1}. \quad (1.139)$$

Proof. We first apply Theorem 6.4.2 of Conn et al. (2000) to deduce that iteration $k-1$ is successful and thus that $x_k = x_{k-1} + s_{k-1} \neq x_{k-1}$. Observe now that the constants $\tilde{\eta}_2$ and κ_{mdc} lie in the interval $(0,1)$, which implies that

$$\frac{(1-\tilde{\eta}_2)}{(3-2\tilde{\eta}_2)} < \frac{1}{2} < 1 \quad \text{and thus} \quad \kappa_{\text{mdc}} \frac{(1-\tilde{\eta}_2)}{(3-2\tilde{\eta}_2)} < 1. \tag{1.140}$$

The conditions (1.138), (1.140), and (1.128), combined with the definition of β_{k-1} in A.4 imply that

$$\Delta_{k-1} < \frac{\|g_{k-1}\|}{\kappa_{\text{ubh}}} < \frac{\|g_{k-1}\|}{\beta_{k-1}}. \tag{1.141}$$

As a consequence, A.4 immediately gives that

$$m_{k-1}(x_{k-1}) - m_{k-1}(x_k) \geq \kappa_{\text{mdc}} \|g_{k-1}\| \min\left[\frac{\|g_{k-1}\|}{\beta_{k-1}}, \Delta_{k-1}\right] = \kappa_{\text{mdc}} \|g_{k-1}\| \Delta_{k-1}. \tag{1.142}$$

On the other hand, we may apply Lemma 1.5.2 and use the triangle inequality to obtain that

$$\begin{aligned} &\big|m_{k-1}(x_{k-1}) - m_{k-1}(x_k)\big| - \big|m_k(x_{k-1}) - m_k(x_k)\big| \\ &\quad \leq \big|\big[m_{k-1}(x_{k-1}) - m_{k-1}(x_k)\big] - \big[m_k(x_{k-1}) - m_k(x_k)\big]\big| \\ &\quad \leq 2\kappa_{\text{ubh}} \Delta_{k-1}^2 \end{aligned}$$

and therefore, with (1.142), that

$$\begin{aligned} |m_k(x_{k-1}) - m_k(x_k)| &\geq |m_{k-1}(x_{k-1}) - m_{k-1}(x_k)| - 2\kappa_{\text{ubh}} \Delta_{k-1}^2 \\ &\geq \kappa_{\text{mdc}} \|g_{k-1}\| \Delta_{k-1} - 2\kappa_{\text{ubh}} \Delta_{k-1}^2. \end{aligned} \tag{1.143}$$

Now (1.138) implies that $(3-2\tilde{\eta}_2)\kappa_{\text{ubh}} \Delta_{k-1} \leq (1-\tilde{\eta}_2)\kappa_{\text{mdc}} \|g_{k-1}\|$ and thus that

$$(1-\tilde{\eta}_2)(\kappa_{\text{mdc}} \|g_{k-1}\| - 2\kappa_{\text{ubh}} \Delta_{k-1}) \geq \kappa_{\text{ubh}} \Delta_{k-1} > 0. \tag{1.144}$$

We finally may apply Theorem 1.5.1 and deduce from A.2, (1.127), (1.143) and (1.144) that

$$|\tilde{\rho}_k - 1| = \left|\frac{f(x_{k-1}) - m_k(x_{k-1})}{m_k(x_{k-1}) - m_k(x_k)}\right| \leq \frac{\kappa_{\text{ubh}} \Delta_{k-1}}{\kappa_{\text{mdc}} \|g_{k-1}\| - 2\kappa_{\text{ubh}} \Delta_{k-1}} \leq 1 - \tilde{\eta}_2. \tag{1.145}$$

Therefore, $\tilde{\rho}_k \geq \tilde{\eta}_2$ and (1.124) then ensures that (1.139) holds. □

It is therefore guaranteed that the trust-region radius can not be decreased indefinitely if the current iterate is not near critically. This is ensured by the next theorem.

Theorem 1.5.4 Suppose that A.1–A.4 hold. Suppose furthermore that there exists a constant κ_{lbg} such that $\|g_k\| \geq \kappa_{\text{lbg}}$ for all k. Then there is a constant κ_{lbd} such that

$$\Delta_k \geq \kappa_{\text{lbd}} \tag{1.146}$$

for all k.

Proof. The proof is the same as for Theorem 6.4.3 in Conn et al. (2000) except that

$$\kappa_{\text{lbd}} = \min\left[1 - \eta_1, \frac{(1-\tilde{\eta}_2)}{(3-2\tilde{\eta}_2)}\right] \frac{\gamma_1 \kappa_{\text{mdc}} \kappa_{\text{lbg}}}{\kappa_{\text{ubh}}}.$$

□

From here on, the proof for the basic trust region applies without change. We first deduce the global convergence of the algorithm to first-order critical points when it generates only finitely many successful iterations.

Theorem 1.5.5 Suppose that A.1–A.4 hold. Suppose furthermore that there are only finitely many successful iterations. Then $x_k = x_*$ for all sufficiently large k and x_* is first-order critical.

Proof. The same argument as in Theorem 6.4.4 in Conn et al. (2000) may be applied since the radius update is identical to that of the basic trust region method for unsuccessful iterations. □

Finally, the next two results ensure the global convergence of the algorithm to first-order critical points, by showing in a first step that at least one accumulation point of the iterates sequence is first-order critical.

Theorem 1.5.6 Suppose that A.1–A.4 hold. Then,

$$\liminf_{k\to\infty} \|\nabla_x f(x_k)\| = 0. \tag{1.147}$$

Proof. See Theorem 6.4.5 in Conn et al. (2000). □

As for the basic trust-region method, this can be extended to show that all limit points are first-order critical.

Theorem 1.5.7 Suppose that A.1–A.4 hold. Then one has that

$$\lim_{k\to\infty} \|\nabla_x f(x_k)\| = 0. \tag{1.148}$$

Proof. See Theorem 6.4.6 in Conn et al. (2000). □

1.5.5.2.2 Convergence to Second-Order Critical Points We now investigate the possibility to exploit second-order information on the objective function, with the aim of ensuring convergence to second-order critical points, i.e. points x_* such that

$$\nabla_x f(x_*) = 0 \quad \text{and} \quad \nabla_{xx} f(x_*) \text{ is positive semidefinite.}$$

Of course, we need to clarify what we precisely mean by "second-order information". We therefore introduce the following additional assumptions:

A.5 The model is asymptotically second-order coherent with the objective function near first-order critical points, i.e.

$$\lim_{k\to\infty} \|\nabla_{xx} f(x_k) - \nabla_{xx} m_k(x_k)\| = 0 \text{ whenever } \lim_{k\to\infty} \|g_k\| = 0.$$

A.6 The Hessian of every model m_k is Lipschitz continuous, that is, there exists a constant κ_{lch} such that, for all k,

$$\|\nabla_{xx} m_k(x) - \nabla_{xx} m_k(y)\| \leq \kappa_{\text{lch}} \|x - y\|$$

for all $x, y \in \mathcal{B}_k$.

A.7 If the smallest eigenvalue τ_k of the Hessian of the model m_k at x_k is negative, then

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{\text{sod}} |\tau_k| \min(\tau_k^2, \Delta_k^2)$$

for some constant $\kappa_{\text{sod}} \in (0, \frac{1}{2})$.

These assumptions are identical to those used in Sections 6.5 and 6.6 of Conn et al. (2000) for the basic trust-region method. The assumption A.6 merely ensures that it is reasonable to impose assumption A.7 (as shown in Theorem 6.6.2 of Conn et al. (2000)). The assumption A.7 says that, if negative curvature appears in the model, if a first-order critical point is approached and the second-order terms of the model appear to be relevant, then this negative curvature will be exploited by the calculation of the trust-region step. Note that the second-order convergence properties of the retrospective trust-region method turn out to be exactly the same as those of the basic trust-region method, and their proofs can essentially been borrowed from this case, with the exception of Lemma 6.5.3. We therefore need to present a proof of that particular result for the new method. As we indicate below, all other results generalize without change and we only mention them for the sake of clarity.

In our analog of Lemma 6.5.3, we assume that the model reduction is eventually significant in the sense that it is at least of the same order as the error between the model and the objective function. We then show that the trust-region radius becomes asymptotically irrelevant if the steps tend to zero.

Lemma 1.5.8 Suppose that A.1–A.3, and A.5 hold. Suppose also that there exists a sequence (k_i) and a constant $\kappa_{\text{mqd}} > 0$ such that

$$m_{k_i}(x_{k_i}) - m_{k_i}(x_{k_i} + s_{k_i}) \geq \kappa_{\text{mqd}} \|s_{k_i}\|^2 > 0 \tag{1.149}$$

for all i sufficiently large. Finally, suppose that

$$\lim_{i \to \infty} \|s_{k_i}\| = 0.$$

Then iteration k_i is successful and

$$\tilde{\rho}_{k_i+1} \geq \tilde{\eta}_2 \quad \text{and} \quad \Delta_{k_i+1} \geq \Delta_{k_i} \tag{1.150}$$

for i sufficiently large.

Proof. We first apply Lemma 6.5.3 of Conn et al. (2000) to deduce that every iteration k_i is successful for i sufficiently large. Now, consider k_i one such iteration. The equations (1.129) and (1.130) imply that for some ξ_{k_i+1} and ζ_{k_i+1} in the segment $[x_{k_i}, x_{k_i+1}]$,

$$\begin{aligned} |\tilde{\rho}_{k_i+1} - 1| &= \left| \frac{f(x_{k_i}) - m_{k_i+1}(x_{k_i})}{m_{k_i+1}(x_{k_i}) - m_{k_i+1}(x_{k_i+1})} \right| \\ &= \left| \frac{\langle s_{k_i}, \nabla_{xx} f(\xi_{k_i+1}) s_{k_i} \rangle - \langle s_{k_i}, \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1}) s_{k_i} \rangle}{- \langle s_{k_i}, g_{k_i+1} \rangle + \frac{1}{2} \langle s_{k_i}, \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1}) s_{k_i} \rangle} \right| \\ &\leq \frac{\|s_{k_i}\|^2 \cdot \|\nabla_{xx} f(\xi_{k_i+1}) - \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1})\|}{\left| - \langle s_{k_i}, g_{k_i+1} \rangle + \frac{1}{2} \langle s_{k_i}, \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1}) s_{k_i} \rangle \right|} \end{aligned} \tag{1.151}$$

where we also used the Cauchy-Schwarz inequality. By substituting $g_{k_i+1} = \nabla_x f(x_{k_i+1})$ (because of A.2) with its expression in (1.136), the denominator D of the latter fraction can be rewritten as

$$D = \left| - \left\langle s_{k_i}, g_{k_i} + \int_0^1 \nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) s_{k_i} \, d\alpha \right\rangle + \frac{1}{2} \langle s_{k_i}, \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1}) s_{k_i} \rangle \right|.$$

Then, replacing $- \langle s_{k_i}, g_{k_i} \rangle$ by its expression in (1.133), we obtain

$$\begin{aligned} D &= \Big| m_{k_i}(x_{k_i}) - m_{k_i}(x_{k_i+1}) + \tfrac{1}{2} \langle s_{k_i}, \nabla_{xx} m_{k_i}(\psi_{k_i}) s_{k_i} \rangle \\ &\quad + \tfrac{1}{2} \langle s_{k_i}, \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1}) s_{k_i} \rangle - \left\langle s_{k_i}, \int_0^1 \nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) s_{k_i} \, d\alpha \right\rangle \Big| \end{aligned}$$

for some ψ_{k_i} in the segment $[x_{k_i}, x_{k_i+1}]$. The triangle inequality, properties of the integral, (1.149), and Cauchy-Schwarz inequality give therefore the following lower bound on D:

$$\begin{aligned} D &\geq |m_{k_i}(x_{k_i}) - m_{k_i}(x_{k_i+1})| \\ &\quad -\tfrac{1}{2}\left| \left\langle s_{k_i}, \int_0^1 [\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} m_{k_i}(\psi_{k_i})] s_{k_i}\, d\alpha \right\rangle \right. \\ &\quad \left. + \left\langle s_{k_i}, \int_0^1 [\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1})] s_{k_i}\, d\alpha \right\rangle \right| \\ &\geq \kappa_{\text{mqd}} \|s_{k_i}\|^2 - \tfrac{1}{2}\|s_{k_i}\| \int_0^1 \|\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} m_{k_i}(\psi_{k_i})\| \cdot \|s_{k_i}\|\, d\alpha \\ &\quad - \tfrac{1}{2}\|s_{k_i}\| \int_0^1 \|\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1})\| \cdot \|s_{k_i}\|\, d\alpha \\ &\geq \|s_{k_i}\|^2 (\kappa_{\text{mqd}} - \tfrac{1}{2}\epsilon_i) \end{aligned} \tag{1.152}$$

where

$$\epsilon_i \stackrel{\text{def}}{=} \int_0^1 \|\nabla_{xx} f(x_{k_i}+\alpha s_{k_i}) - \nabla_{xx} m_{k_i}(\psi_{k_i})\|\, d\alpha + \int_0^1 \|\nabla_{xx} f(x_{k_i}+\alpha s_{k_i}) - \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1})\|\, d\alpha.$$

The triangle inequality now implies that

$$\begin{aligned} \|\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} m_{k_i}(\psi_{k_i})\| &\leq \|\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} f(x_{k_i})\| \\ &\quad + \|\nabla_{xx} f(x_{k_i}) - \nabla_{xx} m_{k_i}(x_{k_i})\| + \|\nabla_{xx} m_{k_i}(x_{k_i}) - \nabla_{xx} m_{k_i}(\psi_{k_i})\| \end{aligned} \tag{1.153}$$

and, similarly, that

$$\begin{aligned} \|\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1})\| &\leq \|\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} f(x_{k_i+1})\| \\ + \|\nabla_{xx} f(x_{k_i+1}) - \nabla_{xx} m_{k_i+1}(x_{k_i+1})\| &+ \|\nabla_{xx} m_{k_i+1}(x_{k_i+1}) - \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1})\|. \end{aligned} \tag{1.154}$$

Since we now observe that

$$\begin{aligned} \|(x_{k_i} + \alpha s_{k_i}) - x_{k_i}\| &\leq \|s_{k_i}\|, & \|\psi_{k_i} - x_{k_i}\| &\leq \|s_{k_i}\|, \\ \|(x_{k_i} + \alpha s_{k_i}) - x_{k_i+1}\| &\leq \|s_{k_i}\|, & \|\zeta_{k_i+1} - x_{k_i+1}\| &\leq \|s_{k_i}\|, \end{aligned}$$

we may deduce that both

$$\|\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} m_{k_i}(\psi_{k_i})\| \quad \text{and} \quad \|\nabla_{xx} f(x_{k_i} + \alpha s_{k_i}) - \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1})\|$$

converge to zero with $\|s_{k_i}\|$ because the first and third terms of the right-hand side of (1.153) and (1.154) tend to zero by continuity of the objective function's and model's Hessians, and because the middle term in the right-hand side of these inequalities also converges to zero because of A.5 and Theorem 1.5.7. As a consequence, $\epsilon_i \leq \kappa_{\text{mqd}}$ when i is sufficiently large, and therefore, combining (1.151) and (1.152), and using the triangle inequality, we obtain

$$\begin{aligned} |\tilde{\rho}_{k_i+1} - 1| &\leq \frac{2}{\kappa_{\text{mqd}}} \|\nabla_{xx} f(\xi_{k_i+1}) - \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1})\| \\ &\leq \frac{2}{\kappa_{\text{mqd}}} \Big[\|\nabla_{xx} f(\xi_{k_i+1}) - \nabla_{xx} f(x_{k_i+1})\| \\ &\qquad + \|\nabla_{xx} f(x_{k_i+1}) - \nabla_{xx} m_{k_i+1}(x_{k_i+1})\| \\ &\qquad + \|\nabla_{xx} m_{k_i+1}(x_{k_i+1}) - \nabla_{xx} m_{k_i+1}(\zeta_{k_i+1})\| \Big] \end{aligned} \tag{1.155}$$

By the same reasoning as for (1.153)–(1.154), the right-hand side of (1.155) tends to zero when i goes to infinity, and $\tilde{\rho}_{k_i+1}$ therefore tends to 1. It is thus larger than $\tilde{\eta}_2 < 1$ for i sufficiently large and (1.150) follows. □

As in Lemma 6.5.4 of Conn et al. (2000), we may apply this result to the entire sequence of iterates and deduce that all iterations are eventually successful and the trust-region radius bounded away from zero.

From here on, the theory in Conn et al. (2000) generalizes without significant change, yielding the following results.

Theorem 1.5.9 Suppose that A.1–A.5 hold and that x_{k_i} is a subsequence of the iterates generated by Algorithm RTR converging to a first-order critical point x_* where the Hessian of the objective function $\nabla_{xx} f(x_*)$ is positive definite. Suppose furthermore that $s_k \neq 0$ for all k sufficiently large. Then the complete sequence of iterates converges to x_*, all iterations are eventually very successful, and the trust-region radius Δ_k is bounded away from zero.

Proof. See Theorem 6.5.5 in Conn et al. (2000). □

We now proof that if the sequence of iterates remains in a compact set, then the existence of at least one second-order critical accumulation point is guaranteed.

Theorem 1.5.10 Suppose that A.1–A.7 hold and that all iterates remain in some compact set. Then there exists at least one limit point x_* of the sequence of iterates x_k produced by Algorithm RTR, which is second-order critical.

Proof. See Theorem 6.6.5 in Conn et al. (2000). □

By just strengthening the radius update rule by requiring that

$$\text{if } \tilde{\rho}_k \geq \tilde{\eta}_2 \text{ and } \Delta_k \leq \Delta_{\max}, \text{ then } \Delta_{k+1} \in [\gamma_3 \Delta_k, \gamma_4 \Delta_k] \tag{1.156}$$

for some $\gamma_4 \geq \gamma_3 > 1$ and some $\Delta_{\max} > 0$, we moreover obtain the second-order criticality of any limit point of the sequence of iterates generated by Algorithm RTR.

Theorem 1.5.11 Suppose that A.1–A.7, and (1.156) hold and let x_* be any limit point of the sequence of iterates. Then x_* is a second-order critical point.

Proof. See Theorem 6.6.8 in Conn et al. (2000). □

Thus the retrospective trust-region algorithm shares all the (interesting) convergence properties of the basic trust-region method under the same assumptions. We conclude this theory section by noting that the above convergence results are still valid if one replaces the Euclidean norm by any (possibly iteration dependent) uniformly equivalent norm, thereby allowing problem scaling and preconditioning.

1.5.5.3 Numerical results

We now consider the numerical behavior of the retrospective algorithm, in comparison with the basic trust-region algorithm BTR.

For the basic algorithm, the trust-region radius update was implemented as in (1.120). To avoid biasing the comparison, we have decided to make as few adaptations as possible to that rule in our retrospective variant (i.e. Step 2 in Algorithm 1.5.2). Thus, if iteration k is unsuccessful, i.e. $\rho_k < \eta_1$ and consequently $x_k = x_{k+1}$, we also decrease the trust-region using the same rule. If, on the contrary, iteration k is successful, i.e. $\rho_k \geq \eta_1$, the trust-region is updated using the procedure described in Step 2 of Algorithm 1.5.2 where we choose the same values for γ_0, γ_1 and γ_2, and take $\tilde{\eta}_1 = \eta_1$ and $\tilde{\eta}_2 = \eta_2$.

We chose to compare the number of iterations to achieve convergence instead of the CPU time or number of function evaluations. Indeed, the cost per iteration is the same for both algorithms and they both evaluate the objective function once per iteration and compute one gradient at every successful iteration. Moreover, timings in Matlab are often difficult to interpret.

Figure 1.6 represents the comparison by a performance profile of the number of iterations of the two algorithms. In this figure, we have only kept the problems for which both algorithms converged to the same local solution (we excluded `BIGGS6`, `BROYDN7D`, `CHAINWOO`, `FLETCHBV`, `LOGHAIRY`, `MEYER3`, `NONCVXU2`, `NONCVXUN`, `SENSORS`, `TOINTGSS` and `VIBRBEAM`). If the subproblem is solved approximately, both algorithms failed on `PALMER1C`, `SBRYBND`, `SCOSINE`, `SCURLY10`, `SCURLY20` and `SCURLY30`. Moreover, RTR failed on `FLETCBV3`, which was solved by BTR. On the other hand, if the subproblem is solved exactly, both algorithms failed on `FLETCBV3` and BTR failed on `SCOSINE`, which was solved by RTR. Note also the number of iterations needed to reach convergence with the RTR algorithm on the highly nonconvex `HUMPS` and `LOGHAIRY` problems is much higher than for the BTR algorithm. The complete numerical results are given in Appendix A.

Our results show that the retrospective algorithm performs as well as the classical one and is just as reliable if the trust-region subproblem is solved approximately. However, if the problem size or structure allows an exact solution, the retrospective algorithm is then significantly more efficient (the improvement is typically of only a few iterations, but is very consistent) and just as reliable. A detailed analysis of our results shows that RTR is in general slightly more conservative than BTR in that it tends to take marginally shorter steps. However, this does not seem to alter performance in a negative way. In particular the longer steps of BTR often result in a larger proportion of unsuccessful iterations (this may be deduced from the result table since the number of unsuccessful iterations is given by the difference between the number of iterations

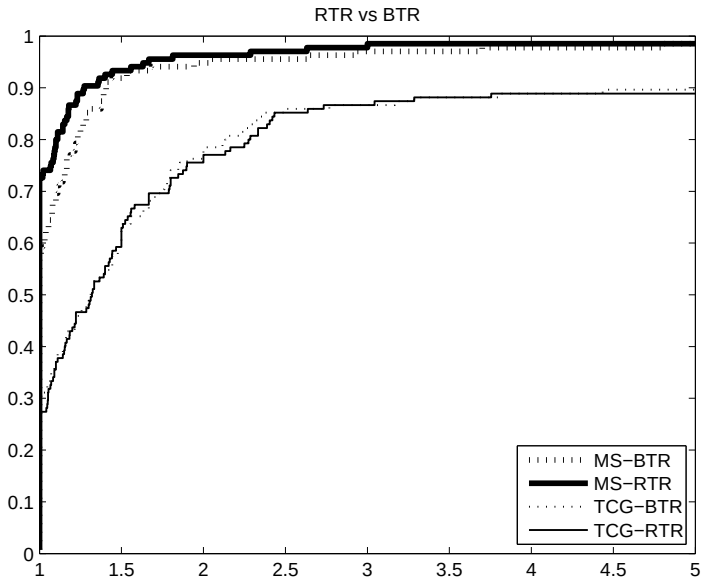

Figure 1.6: Performance profile comparing the number of iterations of the RTR and BTR algorithms

and the number of gradient evaluations). We also note that the choice of an accurate minimization of Newton's model in the trust region also appears to be considerably more efficient than an approximate one, at least in terms of the number of iterations needed for convergence, irrespective of the choice between BTR and RTR. As a consequence, the retrospective variant is clearly at its best when the cost of evaluating the objective function and gradient dominates that of the overall iteration. Additional test not reported here also indicate that both algorithms are essentially indistinguishable when quasi-Newton approximations (SR1 or BFGS) are used instead of the true Hessian. This is perhaps not surprising since the corresponding variants, which use exact solutions of approximate models, may also be interpreted as using approximate solutions of exact models.

Our preliminary numerical experiments indicate that the method is advantageous when the model is good and its quality exploited by an accurate subproblem solution. Moreover this advantage is obtained at essentially zero cost. This new method is especially interesting for adaptive techniques for noisy functions. The potential of the new approach is to exploit the most recent information on the noise to improve numerical performance. Research along this line is ongoing. Other applications of the same idea are also possible across the wide class of trust-region methods, constrained and unconstrained.

Chapter 2

Multilevel optimization

The optimization of finite-dimensional discretizations of problems in infinite dimensional spaces has become a very important area for numerical computations in the last years. New interest in surface design, data assimilation for weather forecasting (Fisher, 1998) or in optimal control of systems described by partial-differential equations have been the main motivation of this challenging research trend, but other applications such as multi-dimensional scaling (Bronstein et al., 2005) or quantization schemes (Emilianenko, 2005) also give rise to similar questions. While the direct solution of such problems for a discretization level yielding the desired accuracy is often possible using existing packages for large-scale numerical optimization, this technique typically does make very little use of the fact that there is an underlying infinite-dimensional problem for which several discretization levels are possible, and the approach thus rapidly becomes cumbersome. This observation motivates the developments of multilevel optimization that makes explicit use of this fact in the hope to allow better efficiency and, possibly, enhance reliability.

Using the different possible levels of discretization for an infinite-dimensional problem is not a new idea. A simple first approach is to use coarser grids in order to compute approximate solutions which can then be used as starting points for the optimization problem on a finer grid (see Griewank and Toint (1982), Bank, Gill and Marcia (2003), Betts and Erb (2003) or Benson, McInnes, Moré and Sarich (2004), for instance). However, potentially more efficient techniques are inspired from the multigrid paradigm in the solution of partial differential equations and associated systems of linear algebraic equations (see, for example Brandt (1977), Bramble (1993), Hackbusch (1994), Hackbusch (1995) or Briggs, Henson and McCormick (2000), for descriptions and references for this much studied topic), and have only been discussed relatively recently in the optimization community.

In this chapter, we first formulate the multilevel optimization problem. Then we present the multigrid methods for linear systems from which the multilevel principles are derived. We then briefly present how these principles could be applied to nonlinear optimization. Then we present two methods that we have constructed using these principles, first the multilevel Moré-Sorensen technique which is a modification of the Moré-Sorensen algorithm presented earlier, and the recursive multilevel trust-region algorithm.

2.1 Position of the problem

Let us consider the multilevel optimization problem

$$\min_{x \in \mathcal{F}} f(x), \tag{2.1}$$

where f is a twice-continuously differentiable objective function which maps $\mathbb{R}^n$ into $\mathbb{R}$ and is bounded below, where $\mathcal{F} = \{x \in \mathbb{R}^n | \, l \leq x \leq u\}$ is a set of bound constraints and where l and u are vectors in $\mathbb{R}^n$ whose entries are possibly infinite.

We assume that a hierarchy of descriptions is known for the considered problem, that is, there exists a collection of functions $\{f_i\}_{i=0}^r$ where each f_i is twice-continuously differentiable and maps $\mathbb{R}^{n_i}$ into $\mathbb{R}$. This type of multilevel structure arises in a variety of forms and applications, but the most common is probably that of discretized infinite-dimensional framework where the f_i represent increasingly finer discretizations of the same infinite-dimensional problem. We suppose that $n_r = n$ and $f_r(x) = f(x)$ for all $x \in \mathbb{R}^n$, giving back our original problem and consider the case where the multilevel hierarchy is useful in the sense that f_i is more costly to minimize than f_{i-1} for each $i = 1, \ldots, r$. To fix terminology, we will refer to a particular i as a *level* and say that level p is *coarser* than level q if $p < q$ and that level p is *finer* than level q if $p > q$. We finally assume that for each level $i = 1, \ldots, r$, there exists a linear full-rank operator R_i from $\mathbb{R}^{n_i}$ to $\mathbb{R}^{n_{i-1}}$ called the *restriction* and another linear full-rank operator P_i from $\mathbb{R}^{n_{i-1}}$ to $\mathbb{R}^{n_i}$ called the *prolongation* and such that

$$\sigma_i P_i = R_i^T, \tag{2.2}$$

for some constant $\sigma_i > 0$ and $\|R_i\|_\infty = 1$.

These assumptions are not new. They are adaptations of a class of methods called multigrid methods used for the solution of partial differential equations and associated systems of linear algebraic equations. Several principles of the methods we have constructed are derived from these methods. Since the understanding of these principles is of great importance to understand the details of our methods, we describe in the next section the method for linear systems.

2.2 Multigrid methods for linear systems

Multigrid methods for linear systems have been developed for more than thirty years. The first multigrid publication is due to Fedorenko (1964) and formulates an algorithm for the discretization of the Poisson equation on a square. But the development of the multigrid methods starts with the pioneering paper of Brandt (1973) where the main principles and the practical utility of multigrid methods are first outlined. Hackbusch (1976) discovered independently the multigrid method and provided reliable methods. We now describe briefly the principles of the multigrid methods.

2.2.1 Discretization on a grid

Let us consider the differential equation

$$\begin{array}{rcll} \mathcal{L}_\Omega u(x) & = & f_\Omega(x) & x \in \Omega, \\ \mathcal{L}_\Gamma u(x) & = & f_\Gamma(x) & x \in \Gamma, \end{array} \tag{2.3}$$

where $\Omega \subset \mathbb{R}^n$ is a given open bounded domain with boundary Γ, $x \in \mathbb{R}^n$, $\mathcal{L}_\Omega$ and $\mathcal{L}_\Gamma$ are differential and boundary operators on respectively Ω and Γ and f_Ω and f_Γ denotes functions defined on respectively Ω and Γ. The solution of (2.3) is denoted by $u(x)$.

For clarity and conciseness, let us consider only a two dimensional problem. We could replace in the previous equation the vector $x \in \mathbb{R}^n$ by a vector denoted (x, y). Dealing with infinite-dimensional problems is impossible on digital computers and consequently, we consider the infinite grid

$$G^h = \{(x, y) \,|\, x = x_i = ih_x,\ y = y_j = jh_y,\ i, j \in \mathbb{Z}\}, \tag{2.4}$$

where $h = (h_x, h_y)$ is a vector of mesh sizes. We define $\Omega^h = \Omega \cap G^h$ and $\Gamma^h = \Gamma \cap G^h$, the intersection of the domain and its boundary with the grid. Instead of considering the infinite dimensional solution $u(x, y)$, we could now only consider the discrete solution u^h defined on $\Omega^h \cup \Gamma^h$. Note that for clarity, instead of $u^h(x, y) = u^h(x_i, y_j)$, we sometimes simply write $u_{i,j}$. We could also denote $\mathcal{L}^h_\Omega$ and $\mathcal{L}^h_\Gamma$ the grid operators. With these definitions, it is possible to consider the discrete analog of (2.3) denoted by

$$\begin{array}{rcll} \mathcal{L}^h_\Omega u^h(x, y) & = & f^h_\Omega(x, y) & (x, y) \in \Omega^h, \\ \mathcal{L}^h_\Gamma u^h(x, y) & = & f^h_\Gamma(x, y) & (x, y) \in \Gamma^h. \end{array} \tag{2.5}$$

Clearly, the concrete definitions of Ω^h, Γ^h, $\mathcal{L}^h_\Omega$ and $\mathcal{L}^h_\Gamma$ depend on the differential equation, on the domain Ω, on the boundary equations and on the discretization, but this introduction is of interest to fix terminology and notation.

To illustrate this derivation of a discretized version of a differential equation, let us consider the Poisson equation,

$$\begin{array}{rcll} -u_{xx} - u_{yy} & = & f(x, y) & (x, y) \in S_2, \\ u & = & 0 & (x, y) \in \partial S_2, \end{array} \tag{2.6}$$

where $S_2 = \{(x, y) \,|\, 0 < x < 1,\ 0 < y < 1\}$ is the two-dimensional unit square. The problem may be cast in a discrete form by defining the square grid

$$G^h = \{(x, y) \,|\, x = x_i = ih_x,\ y = y_j = jh_y,\ i, j = 0 \ldots, n\}, \tag{2.7}$$

where $h_x = h_y = \frac{1}{n}$. To approximate the differential operator, we use the finite difference, that is

$$\begin{array}{rcl} u_{xx}(x_i, y_j) & = & \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2}, \\ u_{yy}(x_i, y_j) & = & \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2}. \end{array} \tag{2.8}$$

Thus, the discretized problem is

$$\begin{array}{l} \frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{h_x^2} + \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{h_y^2} = f_{i,j} \qquad 1 \le i, j \le n-1, \\ u_{i,0} = u_{0,j} = u_{i,n} = u_{n,j} = 0 \end{array} \tag{2.9}$$

where $f_{i,j} = f(x_i, y_j)$. There are thus $(n-1)^2$ interior grid points and the same number of unknowns. This problem is represented in Figure 2.1 where the solid dots indicate the unknowns related to a typical grid point.

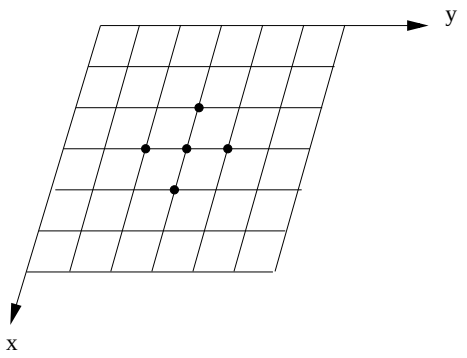

Figure 2.1: Two-dimensional grid for the Poisson problem. Solid dots indicate the unknowns related to a typical grid point.

Let us write $u_i = (u_{i,1}, \ldots, u_{i,n-1})^T$ for $1 \leq i \leq n-1$, the unknowns of the i-th row of the grid and $f_i = (f_{i,1}, \ldots, f_{i,n-1})$, the discretized problem (2.9) may be written as a linear system of the form

$$\frac{1}{h^2}\begin{pmatrix} T & -I & & & & \\ -I & T & -I & & & \\ & -I & T & -I & & \\ & & \ddots & \ddots & \ddots & \\ & & & -I & T & -I \\ & & & & -I & T \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-2} \\ f_{n-1} \end{pmatrix} \tag{2.10}$$

where $h = h_x$, I is the identity matrix of dimension $(n-1, n-1)$ and T is a tridiagonal matrix of dimension $(n-1, n-1)$ of the form

$$T = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}. \tag{2.11}$$

By this simple example, we could see how it is possible to construct from an infinite-dimensional problem a linear system. Even if the solution of the system is possible by standard techniques to solve linear systems, we note that the size of the system may become very large if the number of points of the grid is large and this situation is even truer for three-dimensional problems. Although, it is advantageous to take a large number of discretization points since the accuracy of the finite differences formula used to construct the system depends of the mesh size h. But for very large systems, the standards methods will struggle since they do not consider the

particular structure of the problem. Indeed, even if a discretization has been chosen to construct the linear system, we can even choose another grid with fewer points to construct another linear system and of course there is a link between the solution of these two systems since they are discretizations of the same infinite-dimensional problem. The multigrid method that we present in this section takes the maximum benefit of this structure to obtain the most effective method for this class of problems.

2.2.2 Relaxation methods

To clarify notations, let

$$Au = f \tag{2.12}$$

denote the considered linear system. Consider for simplicity of notation that $u, f \in \mathbb{R}^{n-1}$ and that $A \in \mathbb{R}^{(n-1)\times(n-1)}$. Let us denote u the exact solution of the system and v an approximation to it, perhaps generated by an iterative method. Let us define the error given by

$$e \stackrel{\text{def}}{=} u - v, \tag{2.13}$$

which is simply the difference between the approximation and the exact solution. The error is inaccessible as the exact solution. However, a measure of how well v approximates u is the residual given by

$$r \stackrel{\text{def}}{=} f - Av. \tag{2.14}$$

The residual is the amount by which the approximation v fails to satisfy the original problem (2.12). If the system admits a unique solution, then $r = 0$ if and only if $e = 0$. However, this may not be true that when r is small in norm, e is also small in norm.

By using the definitions of the error and of the residual, we can derive a very important relationship between the error and the residual, called the residual equation and given by

$$Ae = r. \tag{2.15}$$

Suppose that an approximation v has been computed by some method. It is easy to compute the residual and to improve the approximation v, we might solve the residual equation to obtain an approximate e and then compute the new approximation by

$$v_{\text{new}} = v + e. \tag{2.16}$$

This two-stages strategy is used in the multigrid methods. But before introducing the multigrid method, we have to introduce the main ingredient of the first stage of this strategy, namely the relaxation methods.

The commonly used relaxation schemes are particular cases of methods called stationary linear methods. These methods iteratively improve an approximation v of the system solution by applying

$$v_m = Rv_{m-1} + Bf, \tag{2.17}$$

where v_i denotes the approximation at iteration i, B is a matrix approximating the inverse of the system matrix A^{-1} and $R = I - BA$ is the general iteration matrix. Let us decompose the system matrix A as

$$A = D - L - U, \tag{2.18}$$

where D is the diagonal , $-L$ the strictly lower part and $-U$ the strictly upper part of matrix A. The most used methods of this type are

the Jacobi method:

$$v_m = R_J v_{m-1} + D^{-1} f, \tag{2.19}$$

where $R_j = D^{-1}(L + U)$.

the Gauss-Seidel method:

$$v_m = R_G v_{m-1} + (D - L)^{-1} f, \tag{2.20}$$

where $R_G = (D - L)^{-1} U$.

These methods may appear abstract but their definitions just result of modifications of (2.12). Indeed, if we consider for example the Gauss-Seidel method, then

$$\begin{array}{rrcl} & Au & = & f \\ \iff & (D - L - U)u & = & f \\ \iff & (D - L)u & = & Uu + f \\ \iff & u & = & (D - L)^{-1} U u + (D - L)^{-1} f \\ \iff & u & = & R_G u + (D - L)^{-1} f \end{array} \tag{2.21}$$

The main difference between the Gauss-Seidel method and the Jacobi method is that the former uses the most recent information. Indeed, if we write the method componentwise we have,

$$v_{m,j} = \frac{f_j - \sum_{i=0}^{j-1} A_{ji} v_{m,i} - \sum_{i=j+1}^{n} A_{ji} v_{m-1,i}}{A_{jj}}, \tag{2.22}$$

where $v_{k,i}$ denotes the i-th component of the k-th approximation v_k. We note that for the component j of vector v_m, we already use the components already computed $v_{m,1}, \ldots, v_{m,j-1}$.

These methods may be rewritten as

$$v_m = R v_{m-1} + g, \tag{2.23}$$

and moreover these methods are such that the exact solution u is a fixed point of the iteration, that is

$$u = Ru + g. \tag{2.24}$$

By subtracting these last two equations, we have

$$e_m = R e_{m-1} \tag{2.25}$$

linking the error of two successive iterations. And by applying this equation recursively, we have that

$$e_m = R^m e_0. \tag{2.26}$$

And by norm properties, it is possible to bound the error after m iterations,

$$\|e_m\| \leq \|R\|^m \|e_0\|. \tag{2.27}$$

Consequently, the method associated to the matrix R converges for every starting point if and only if $\rho(R) < 1$, where $\rho(A)$ denotes the spectral radius of matrix A, that is $\rho(A) = \max_i |\lambda_i|$ where λ_i are the eigenvalues of matrix A. Thus, if for a given problem, all eigenvalues of the matrix R are smaller than 1 (in magnitude), the method converges.

But we can go further when considering the convergence of the relaxation methods. Indeed, it is possible to decompose the error into sinusoidal functions

$$e_0 = \sum_{k=1}^{n-1} c_k \sin(k\frac{2\pi}{n}) \tag{2.28}$$

where $c_k \in \mathbb{R}$. The functions $\sin(k\frac{2\pi}{n})$ are called the Fourier modes and they are split into two categories, the smooth modes when $1 \leq k < \frac{n}{2}$ and the oscillatory modes when $\frac{n}{2} \leq k \leq n-1$. It is possible to show using local Fourier analysis that the relaxation methods effectively reduce the oscillatory modes of the error but have difficulties to decrease the smooth modes of the error. This fact is quite technical to show and depends of the method used and of the considered problem (but the interested reader may see Trottenberg, Oosterlee and Schüller (2001) or Wesseling (1992)). However, this fact is illustrated on Figure 2.2 where the Gauss-Seidel method is applied on problem (2.10).

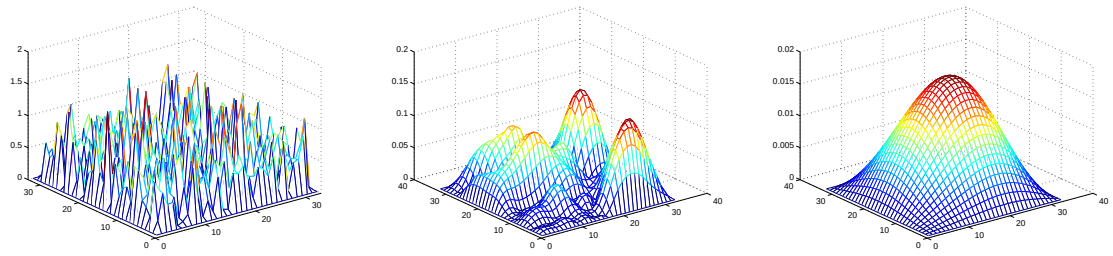

Figure 2.2: Evolution of the error when the Gauss-Seidel method is applied to the Poisson problem. The figure on the left represents the initial error and the figures on the center and on the right represents the error respectively after 10 and 100 iterations. The scale is each time reduced by a factor 10.

Many relaxation schemes possess this property called smoothing property which is a serious limitation. However, this limitation can be overcome and the remedy is one of the pathways to multigrid.

2.2.3 Coarser representations

One way to improve the relaxation methods is to use the coarser representations of the problem. This is the improvement chosen in the multigrid methods. This concept is quite simple.

Assume that some iterations of a relaxation method have been applied to a one-dimensional problem discretized on a grid G^h until only smooth error modes remain. We represent these components on a coarser grid G^{2h}. An example is illustrated in Figure 2.3. We immediately see that the smooth mode appear more oscillatory when represented on the coarse grid.

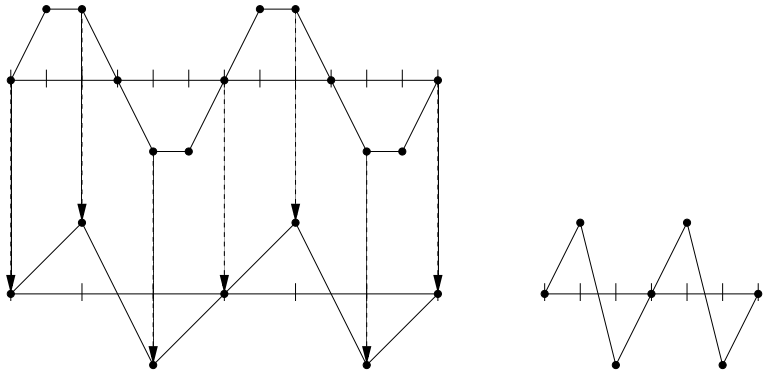

Figure 2.3: The left picture represents a smooth mode of an error projected on a coarser grid. The right picture represents the left coarser grid but with the mesh scale identical to the left finer grid to see directly that the mode appears more oscillatory.

This could be explained easily. We have defined the Fourier modes of the error as the functions $\sin(k\frac{2\pi}{n})$ and say that the smooth modes correspond to the functions where $1 \leq k < \frac{n}{2}$. If we consider now the problem discretized on the coarser grid, the number of unknowns n_C is $\frac{n-1}{2}$. Thus the smooth modes on the fine grid for which k is near $\frac{n}{2}$ are oscillatory on the coarser grid. It is important to note that on the contrary, oscillatory modes of the fine grid may not be well represented on the coarser grid and thus have to be eliminated directly on the fine grid. Thus, after some iterations of a relaxation procedure, we have to eliminate the fine smooth modes of the error. We represent these modes on the coarse grid on which they appear more oscillatory. Consequently, the same relaxation methods may be applied on the coarse grid. This is more efficient since the modes are more oscillatory and cheaper since the number of variables is smaller.

Another strategy to improve the relaxation methods is to start from better initial guess. And again, we use the coarser representations to improve our starting point. Indeed, we could use the relaxation schemes on very coarse representations of the problem and using as a starting guess for a finer representation the solution of the problem on the coarse grid. We use this strategy until we reach the considered discretization hopefully with a good initial approximation. This strategy is called nested iteration or mesh refinement strategy

Of course a lot of details are left unknown after the explanation of these two strategies to improve the relaxation methods. The most important is surely the mechanism we use to transfer information from a coarse level to the finer level or from the fine level to the coarse level. This is done by the transfer operators explained at the next section. These operators allow us also to define the original problem on a coarse grid.

2.2.4 Transfer operators

Let us consider a one-dimensional problem. We assume that the mesh size of the coarse grid is twice the mesh size of the fine grid. This is a common practice since there is no advantage to take mesh sizes with ratios other than 2 (see Briggs et al. (2000)). Let us first consider the case where we want to represent a quantity on the fine level from a quantity on the coarse level. This strategy is common in numerical analysis and is called interpolation or prolongation. Many interpolation methods may be used. However, in practice, the simplest strategies are quite effective. This is why we consider linear interpolation.

Let us denote v^{2h} a coarse quantity, the 1-dimensional linear interpolation operator is denoted by I_{2h}^h and produces fine-grid vectors according to the rule $I_{2h}^h v^{2h} = v^h$, where

$$\begin{array}{rcl} v_{2j}^h & = & v_j^{2h}, \\ v_{2j+1}^h & = & \frac{1}{2}(v_j^{2h} + v_{j+1}^{2h}), \end{array} \quad 0 \leq j \leq \frac{n}{2} - 1. \tag{2.29}$$

It means that each fine-grid component which is also a coarse-grid component is exactly the coarse-grid component and each fine-grid component which is not a coarse-grid component is the arithmetic mean of its coarse neighbors. Thus, the operator has full-rank and is defined by the matrix,

$$I_{2h}^h = \frac{1}{2} \begin{pmatrix} 1 & & & \\ 2 & & & \\ 1 & 1 & & \\ & 2 & & \\ & 1 & \ddots & 1 \\ & & & 2 \\ & & & 1 \end{pmatrix}. \tag{2.30}$$

The interpolation operator is represented in Figure 2.4. It is also possible to define linear interpolation operator for 2-dimensional problems. To define it, we first need to define the Kronecker product of two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$ which is the matrix $A \otimes B \in \mathbb{R}^{mp \times nq}$ such that

$$A \otimes B \stackrel{\text{def}}{=} \begin{pmatrix} A_{11}B & \dots & A_{1n}B \\ \vdots & \ddots & \vdots \\ A_{m1}B & \dots & A_{mn}B \end{pmatrix}, \tag{2.31}$$

where $A_{ij}B$ is the matrix $C \in \mathbb{R}^{p \times q}$ such that $C_{kl} = A_{ij}B_{kl}$. The 2-dimensional interpolation operator J_{2h}^h is then simply,

$$J_{2h}^h = I_{2h}^h \otimes I_{2h}^h. \tag{2.32}$$

Now, we consider the second class of intergrid operators where we want to represent a quantity on the coarse level from a quantity on the fine level. These class of operators are called restriction operators and are denoted by I_h^{2h}. The most obvious operator is injection where each coarse-grid component is simply the associated fine grid component. However, practitioners prefer using the full weighting operator defined by $I_h^{2h} v^h = v^{2h}$, where

$$v_j^{2h} = \frac{1}{4}(v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h), \quad 1 \leq j \leq \frac{n}{2} - 1. \tag{2.33}$$

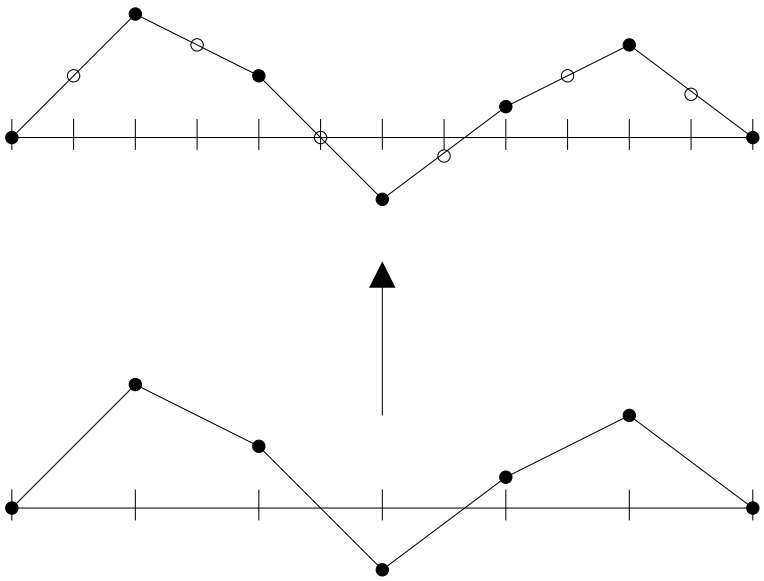

Figure 2.4: Description of a prolongation operator.

Thus, we could write $I_h^{2h} = \frac{1}{2}(I_{2h}^h)^T$ which is the variational property

$$I_h^{2h} = c(I_{2h}^h)^T, \quad c \in \mathbb{R}, \tag{2.34}$$

which is very important for the convergence demonstration of the multigrid methods. The restriction operator is represented in Figure 2.5. It is also possible to define restriction operator in greater dimensions. The 2-dimensional restriction operator J_h^{2h} is defined by

$$J_h^{2h} = I_h^{2h} \otimes I_h^{2h}. \tag{2.35}$$

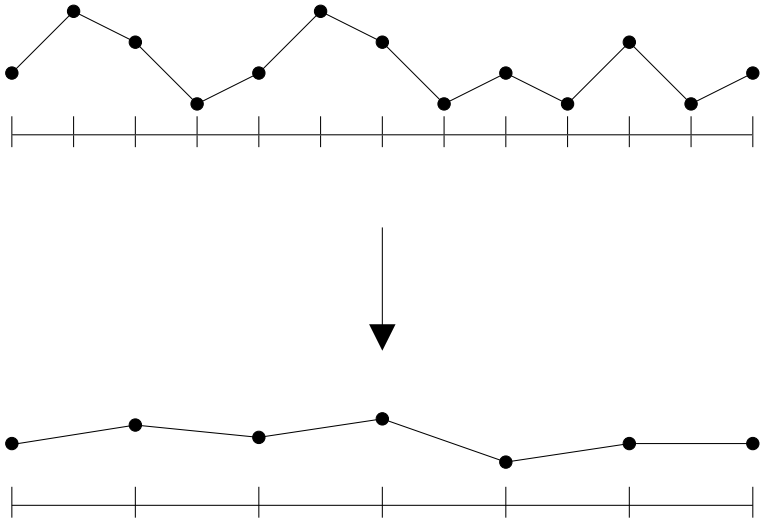

Figure 2.5: Description of a restriction operator.

Now that the transfer operators are defined, it is possible to describe precisely the multigrid schemes and to describe the definition of the coarse problem.

2.2.5 Multigrid schemes

Let us first consider the definition of the coarse level problem. Let us remind the residual equation on the fine level

$$A^h e^h = r^h. \tag{2.36}$$

We want to define a coarse representation of this equation. Of course, we could discretize the infinite-dimensional problem on a grid with mesh size $2h$ and derive the new equation. But multigrid practitioners prefer defining the equation in an other way.

Assume that the error e^h lies in the range of the prolongation operator I_{2h}^h, we could write $e^h = I_{2h}^h u^{2h}$ for some vector $u^{2h} \in \Omega^{2h}$. In this case, the residual equation (2.36) may be written as

$$A^h I_{2h}^h u^{2h} = r^h. \tag{2.37}$$

Now, if we analyze the operator $A^h I_{2h}^h$, we may show that its odd rows are zero and that its even rows correspond to the coarse-grid points of Ω^{2h}. Therefore, to derive the coarse residual equation, we can drop the odd rows of the operator by applying the restriction operator I_h^{2h}. The coarse residual equation is then

$$I_h^{2h} A^h I_{2h}^h u^{2h} = I_h^{2h} r^h, \tag{2.38}$$

and we will denote the coarse grid operator

$$A^{2h} \stackrel{\text{def}}{=} I_h^{2h} A^h I_{2h}^h. \tag{2.39}$$

This operator is called the *Galerkin operator*. This operator is not the only possible choice. However, it has many interesting properties, such as keeping the coarse level operator symmetric and positive definite, if that is the case for the fine one, and maintaining the sparsity created by the discretization.

We have now developed all the ingredients needed to explain the multigrid algorithm which is given by Algorithm 2.2.1.

The parameter ν controls the number of relaxation iterations before and after visiting the coarse grid. Numerical experiments show that a value between 1 and 3 provides good results.

This procedure can be applied recursively, in that the solution of the residual equation in the coarse level itself can be computed recursively. At the coarsest level, which corresponds to the smallest system and where recursion is no longer possible, the solution may be computed exactly, for instance by using matrix factorization or we could apply a relaxation scheme. This recursive strategy is formalized in Algorithm 2.2.2.

This algorithm goes down to the coarsest grid and then goes back to the finest grid. This mechanism is represented in Figure 2.6 and because of its pattern, this algorithm is called V-cycle. It is possible to define more complicated algorithms by applying μ times the last equation of (2.40). This strategy is then called the μ-cycle scheme but in practice, only $\mu = 1$ (which

Algorithm 2.2.1: Multigrid correction scheme

v^h = **MG(**v^h, A^h, f^h**)**

Step 1. Apply ν iterations of a relaxation scheme on problem $A^h u^h = f^h$ with initial guess v^h.

Step 2. Compute the fine-grid residual $r^h = f^h - A^h v^h$. Compute the coarse matrix $A^{2h} = I_h^{2h} A^h I_{2h}^h$ and coarse residual $r^{2h} = I_h^{2h} r^h$.

Step 3. Solve $A^{2h} e^{2h} = r^{2h}$.

Step 4. Prolongate the coarse grid error by $e^h = I_{2h}^h e^{2h}$ and correct the approximation $v^h = v^h + e^h$.

Step 5. Apply ν iterations of a relaxation scheme on problem $A^h u^h = f^h$ with initial guess v^h.

gives the V-cycle) and $\mu = 2$ are used. The strategy with $\mu = 2$ is called W-cycle and is represented also in Figure 2.6.

Before introducing the intergrid operators, we have stated that there were two adaptations to improve the relaxation scheme. We have introduced the correction scheme and we now introduce the second way, the mesh-refinement process. As said before, the idea is to start from the coarsest level and to solve successively the system using as starting point the prolongated solution of the coarser level. But now, for the solution of the system, we use the Algorithm 2.2.2 to take full advantage of the level structure. This strategy leads to the Full-Multigrid Algorithm which is formalized in Algorithm 2.2.3. The mechanism is represented in Figure 2.7

The Full-Multigrid algorithm is particularly efficient for linear systems of equations which are discretization of elliptical differential equations. A robust convergence theory may be shown but is not reported here (the interested reader may see Trottenberg et al. (2001) or Wesseling (1992) or Briggs et al. (2000)). Note also that this introduction is written for problems which are discretizations of infinite-dimensional problems or irregular grids but it is possible to apply multigrid methods on problems which are not defined on a grid. These methods are called algebraic multigrid methods where the same ideas are applied. We finish this small introduction on multigrid techniques by representing the results obtained by the V-cycle algorithm on the same example than Figure 2.2. The results are presented in Figure 2.8 where the figure scale is the same as the one used in Figure 2.2. As we can see, the multigrid method outperforms the relaxation methods. We describe in the next section how these multigrid ideas may be applied to nonlinear optimization.

Algorithm 2.2.2: V-cycle scheme

v^h = **V(**v^h, A^h, f^h**)**

Step 1. Apply ν iterations of a relaxation scheme on problem $A^h u^h = f^h$ with initial guess v^h.

Step 2. If we are on the coarsest level, go to Step 4. Else, apply the algorithm recursively by

$$\begin{aligned} f^{2h} &= I_h^{2h}(f^h - A^h v^h), \\ A^{2h} &= I_h^{2h} A^h I_{2h}^h, \\ v^{2h} &= 0, \\ v^{2h} &= V(v^{2h}, A^{2h}, f^{2h}). \end{aligned} \tag{2.40}$$

Step 3. Correct the approximation $v^h = v^h + I_{2h}^h v^{2h}$.

Step 4. Apply ν iterations of a relaxation scheme on problem $A^h u^h = f^h$ with initial guess v^h.

2.3 Application of the multigrid principles to optimization

The application of the multigrid principles is clearly of interest. A lot of applications in surface design, data assimilation for weather forecasting (Fisher, 1998) or in optimal control of systems described by partial-differential equations have been the main motivation of this challenging research trend, but other applications such as multi-dimensional scaling (Bronstein et al., 2005) or quantization schemes (Emilianenko, 2005) also give rise to similar questions.

The optimization community has applied the multigrid principles to all the globalization schemes presented in Chapter 1. We briefly present in this section how people have adapted the methods before introducing more in details, the methods we have constructed.

2.3.1 Multilevel linesearch methods

The first application of the multilevel philosophy was in linesearch-based optimization. The methods developed by Fisher (1998) or Nash (2000) and Lewis and Nash (2005) use two different descent directions. The first one is a classical one and the second one is a direction obtained using the coarse definition of the problem. The method is defined recursively, using coarser definitions of the problem if any to compute the coarse direction. The method alternates between the classical and coarse directions combining effectively the elimination of the smooth and oscillatory components of the error. The method obtained is numerically efficient and is proved to converge for convex optimization problems in the case of elliptic partial differential operators (see Borzi and Kunisch (2006)).

Another method developed by Wen and Goldfarb (2008) uses a descent condition restricting the use of the coarser directions to the cases where the approximate decrease using the coarser

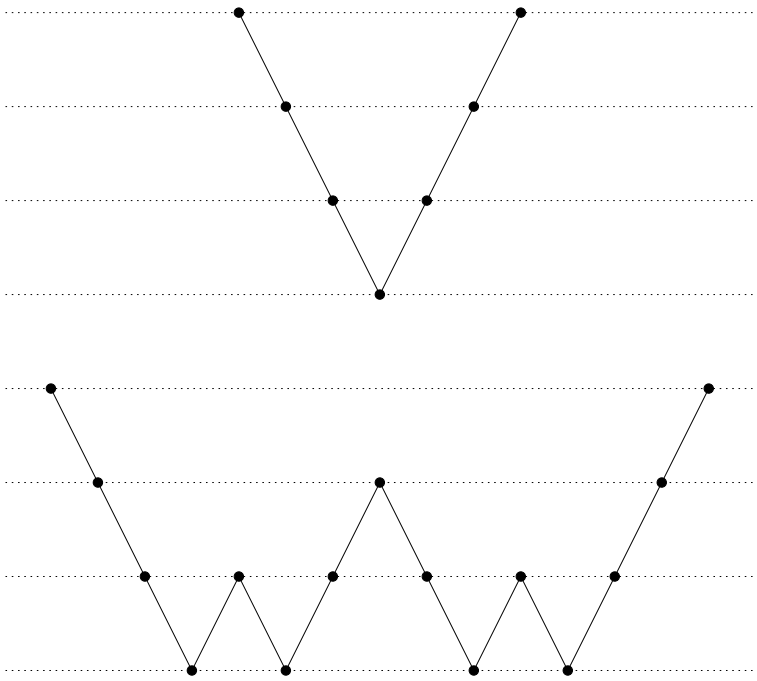

Figure 2.6: The above picture represents a V-cycle and the under picture a W-cycle.

direction (which is computationally more effective) is comparable to the approximate decrease using the classical direction (as first proposed by Gratton, Sartenaer and Toint (2008*b*)). This condition allows them to prove the convergence of the method for nonconvex optimization problems.

2.3.2 Multilevel adaptive cubic overestimation technique

The adaptive cubic overestimation technique proposed by Cartis et al. (2009) provides better numerical results than the other globalization methods for general optimization problems. We have thus logically apply the multilevel philosophy to this method.

In our method, we construct two different models of the objective function, a coarse local model using the coarse definitions of the optimization problems and the classical third-order model. We then use the descent condition proposed by Gratton et al. (2008*b*) to choose between the two models. If the coarse local model is chosen, the algorithm is called recursively. Otherwise, we apply a modification of the Gauss-Seidel method to minimize the third-order model.

This method found the solution of our example problems (both convex and nonconvex), but it is numerically not efficient compared to the other multilevel optimization methods. Indeed, in the other multilevel strategies, the adapted relaxation scheme is of low cost (typically $\mathcal{O}(np)$ where n is the number of variables and p the bandwidth of the Hessian). But to minimize this third-order model, the cost of the adaptations of the existing smoothing technique is larger (typically $\mathcal{O}(n^2)$). Thus, the method obtained, although requiring less iterations and thus less function evaluations, is slower for our test problems. Although, the method may be interesting

Algorithm 2.2.3: Full-Multigrid V-cycle

v^h **= FMG(**v^h, A^h, f^h**)**

Step 1. If we are on the coarsest level, set $v^h = 0$ and go to Step 3. Else, apply the algorithm recursively by

$$\begin{aligned} f^{2h} &= I_h^{2h} f^h, \\ A^{2h} &= I_h^{2h} A^h I_{2h}^h, \\ v^{2h} &= FMG(v^{2h}, A^{2h}, f^{2h}). \end{aligned} \tag{2.41}$$

Step 2. Compute the starting point $v^h = I_{2h}^h v^{2h}$.

Step 3. Solve the system using v^h = V(v^h, A^h, f^h).

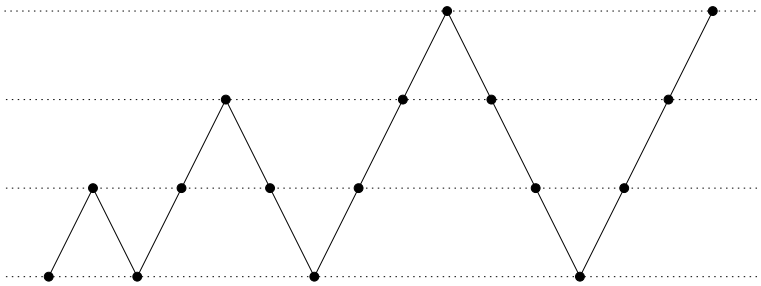

Figure 2.7: Representation of the Full Multigrid algorithm.

in the case of very costly function evaluations.

2.3.3 Multilevel trust-region techniques

The first multilevel trust-region method was introduced by Gratton et al. (2008*b*). In this method, two models are constructed, a coarse local model using the coarse definitions of the optimization problems and a second-order Taylor model. The model to be minimized is chosen by the mean of a descent direction comparing the approximate possible decrease obtained by the models. The chosen model is then minimized in an Euclidean trust-region and then the trial point is accepted or rejected and the trust-region radius is updated in a classical trust-region framework.

In a more recent paper, Gratton, Mouffe, Toint and Weber-Mendonca (2008*a*) modify the first method using the infinity-norm to define the recursive trust regions, which results in substantial algorithmic simplifications when compared to the earlier algorithm using the Euclidean norm[(1)]. It also allows the incorporation of bound constraints into the problem, a feature missing in the earlier version. This method is presented more in details at the end of the chapter.

[(1)]We refer the reader to Gratton et al. (2008*a*) for a more complete discussion of these advantages.

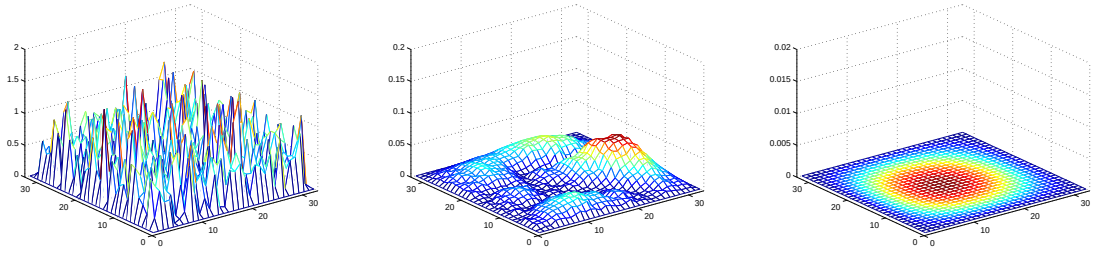

Figure 2.8: Evolution of the error when the V-cycle method is applied to the Poisson problem. The figure on the left represents the initial error and the figures on the center and on the right represents the error respectively after 10 and 100 iterations. The scale is the same as the one used in Figure 2.2.

An other way to consider the application of the multigrid concepts to the trust-region method is to apply the multigrid techniques for the (exact) solution of the trust-region subproblem at the highest level, that is for the finest discretization. If the objective function is (locally) convex, then a suitable optimizing step is derived from the solution of (a variant of) Newton's equations (see Theorem 1.4.1), which often results in solving a positive definite linear system. This is for instance the case if the local Hessian is given by a discretized Laplacian or other elliptic operator. In this case, we can very naturally consider applying a multigrid linear solver to this system, yielding a very efficient method to compute the step. However, things become much less clear when the objective function is locally non-convex, in which case a suitable step is no longer given by Newton's equations. The Moré-Sorensen algorithm (see Algorithm 1.4.2) may be applied for computing a step in this case for small dimensional problems and guarantee, in most cases, that every limit point of the sequence of iterates is a second-order stationary point. However, this method is unfortunately very often impractical for large discretized problems because it involves factorizing a Hessian matrix defined on the fine mesh. This is particularly true if we consider the discretization of variational problems in three dimensions or more. We thus have constructed two variants of the Moré-Sorensen algorithm that are suitable for these large problems but nevertheless guarantee convergence to second-order limit points. These variants are again constructed using the multigrid principle and are presented in the next section.

2.4 The multilevel Moré-Sorensen method

2.4.1 Description of the method

We first remind some notations of the first Chapter. We consider the solution of an unconstrained optimization problem of the form

$$\min_{x \in \mathbb{R}^n} f(x), \qquad f : \mathbb{R}^n \to \mathbb{R}, \tag{2.42}$$

where f is a twice continuously differentiable function and bounded below. The standard *trust-region subproblem* defining s_k is then to solve

$$\min_{\|s\| \leq \Delta} m_k(x_k + s) \stackrel{\text{def}}{=} \langle g_k, s \rangle + \tfrac{1}{2} \langle s, H_k s \rangle, \tag{2.43}$$

where $g_k = \nabla_x f(x_k)$ and H_k is a bounded symmetric approximation of $\nabla_{xx} f(x_k)$. We focus here on the Euclidean norm for defining the trust region because it has the remarkable property that the solution of (2.43) is computationally tractable even if H_k is indefinite (Vavasis and Zippel (1990)). Approximate solutions of this subproblem may also be computed by applying iterative methods such as the conjugate-gradients or generalized Lanczos-trust-region algorithms, but we focus here on the exact solution of (2.43).

Theorem 1.4.1 states that any global minimizer s^M of (2.43) satisfies the system of linear equations[(2)]

$$H(\lambda^M) s^M = -g, \tag{2.44}$$

where $H(\lambda^M) \stackrel{\text{def}}{=} H + \lambda^M I$ is positive semidefinite, $\lambda^M \geq 0$ and $\lambda^M(\|s^M\| - \Delta) = 0$, with s^M being unique if $H(\lambda^M)$ is positive definite. This result indicates that s^M can be seen as the unconstrained minimizer of a quadratic model whose Hessian is made sufficiently positive definite by adding the term $\lambda^M I$. The Hessian curvature induced by this additional term must thus be strong enough to force s^M to lie within the trust region.

We also know that, if $\lambda > -\lambda_{\min}(H)$, where $\lambda_{\min}$ is the smallest eigenvalue of H, then $H(\lambda)$ is positive definite and the system (2.44) has a unique solution,

$$s(\lambda) = -H(\lambda)^{-1} g, \tag{2.45}$$

which must satisfy the nonlinear inequality

$$\|s(\lambda)\| \leq \Delta \tag{2.46}$$

whenever $\lambda = 0$ or the nonlinear equality

$$\|s(\lambda)\| = \Delta \tag{2.47}$$

if $\lambda > 0$.

We now wish to develop an algorithm for the solution of (2.42) that follows the general pattern of the Moré-Sorensen method but which, at the same time, exploits the ideas and techniques of multigrid. If the problem is convex and the multiplier λ^* is known, we propose to use a multigrid solver for the system (2.44), thereby exploiting the hierarchy of level-dependent problem formulations. If the multiplier is not known, we also face, as in the standard Moré-Sorensen method, the task to find its value, again exploiting the multilevel nature of the problem.

Let us denote $R_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i-1}}$ and $P_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ for $i = 1, \ldots, r$ (the *restriction* and the *prolongation*, respectively) such that $P_i = \sigma_i R_i^T$, with $\sigma_i > 0$, for all $i = 1, \ldots, r$. We will call each i a *level*, with $n_r = n$ such that $H_r(\lambda) = H(\lambda)$. In this case, we can construct a

[(2)] In what follows, since we will describe what happens within a single "outer" trust-region iteration, we will drop the iteration indexes k for simplicity.

simpler representation of the matrix as the *Galerkin operator* (see Trottenberg et al. (2001) for a description of the operator and its properties) for $H_i(\lambda)$ defined by

$$H_{i-1}(\lambda) = R_i H_i(\lambda) P_i. \tag{2.48}$$

These definitions allow us to apply the multigrid algorithm to (2.44), and in addition, we must, as in Algorithm 1.4.2, find a new value of λ if the step computed as the solution of (2.44) does not satisfy our stopping conditions. Finding the value of λ^* may in practice be considered as a two-stages process. We first need to find a lower bound $\lambda^L \geq 0$ such that $H_r(\lambda)$ is positive semidefinite for all $\lambda \geq \lambda^L$. Assuming that $\lambda^* = 0$ does not solve the problem (in the sense of (2.46)), the second is then to determine $\lambda^* \geq \lambda^L$ such that

$$\|s_r(\lambda^*)\|_2 = \|H_r(\lambda^*)^{-1} g\|_2 = \Delta, \tag{2.49}$$

where we have simply rewritten (2.45) and (2.47) at level r, the topmost in our hierarchy. In our multigrid context, we intend to exploit the restriction of that problem on the i-th level where

$$\|s_i(\lambda^*)\|_i = \|H_i(\lambda^*)^{-1} g_i\|_i = \Delta, \tag{2.50}$$

where,

$$M_i \stackrel{\text{def}}{=} \prod_{\ell=i+1}^{r} R_\ell, \quad Q_i \stackrel{\text{def}}{=} \prod_{\ell=r}^{i+1} P_\ell, \quad g_i = M_i g \quad \text{and} \quad \|x\|_i \stackrel{\text{def}}{=} \|Q_i x\|_2.$$

The linear system implicit in (2.50) is then solved using the multigrid technique. But note that at the opposite of the multigrid methods that always use the coarse representations, here, if the norm of the restricted residual of the system is not large enough compared with the norm of the residual at level i, i.e. if $\|r_{i-1,0}\| < \kappa_{\text{r}} \|r_{i,k}\|$ for some $\kappa_{\text{r}} < 1$, then there is no advantage in trying to solve the lower level system. In this case, we perform smoothing iterations similar to those used in classical multigrid methods. Otherwise, if

$$\|r_{i-1,0}\| \geq \kappa_{\text{r}} \|r_{i,k}\|, \tag{2.51}$$

we compute a solution of the lower level residual equation.

2.4.1.1 Exploiting the Level Structure to Find Bounds on λ^*

Consider ensuring positive semidefiniteness of $H_r(\lambda)$ first. Our structure exploiting approach for this question is based on the simple observation that $H_i(\lambda)$ ($i = 2, \ldots, r$) cannot be positive semidefinite if $H_{i-1}(\lambda)$ is not, as expressed by the following property.

Lemma 2.4.1 *Let $P_i \in \mathbb{R}^{n_i \times n_{i-1}}$ be a full (column) rank matrix. If $\lambda_1^i \leq \ldots \leq \lambda_{n_i}^i$ are the eigenvalues of $A \in \mathbb{R}^{n_i \times n_i}$, and $\lambda_1^{i-1} \leq \ldots \leq \lambda_{n_{i-1}}^{i-1}$ are the eigenvalues of $R_i A P_i \in \mathbb{R}^{n_{i-1} \times n_{i-1}}$, where $R_i = \frac{1}{\sigma_i} P_i^T$ for some $\sigma_i > 0$, then*

$$\lambda_1^{i-1} \geq \frac{\kappa_{\min}^2}{\sigma_i} \lambda_1^i, \tag{2.52}$$

where $\kappa_{\min}$ is the smallest singular value of P_i.

Proof. Using the extremal properties of eigenvalues (see Golub and Van Loan (1983)), we see that

$$\lambda_1^{i-1} = \min_{\substack{x\in\mathbb{R}^{n_{i-1}}\\ \|x\|_2=1}} \frac{\langle x, P_i^T A P_i x\rangle}{\sigma_i} = \min_{\substack{x\in\mathbb{R}^{n_{i-1}}\\ \|x\|_2=1}} \frac{\langle P_i x, A P_i x\rangle}{\sigma_i} = \min_{\substack{y=P_i x\\ \|x\|_2=1}} \frac{\langle y, Ay\rangle}{\sigma_i}.$$

But, since $\|y\|_2 = \|P_i x\|_2 \geq \kappa_{\min}$, we obtain that

$$\lambda_1^{i-1} = \min_{\substack{y=P_i x\\ \|x\|_2=1}} \frac{\kappa_{\min}^2\langle y, Ay\rangle}{\sigma_i\kappa_{\min}^2} \geq \min_{\substack{y=P_i x\\ \|x\|_2=1}} \frac{\kappa_{\min}^2\langle y, Ay\rangle}{\sigma_i\|y\|_2^2} \geq \min_{y\in\mathbb{R}^{n_i}} \frac{\kappa_{\min}^2\langle y, Ay\rangle}{\sigma_i\|y\|_2^2} = \frac{\kappa_{\min}^2}{\sigma_i}\lambda_1^i.$$

□

This property thus implies that the value of the multiplier needed to make $H_{i-1}(\lambda)$ convex provides a computable lower bound on that needed to make $H_i(\lambda)$ convex. In many cases of interest, the value of $\kappa_{\min}$ is known and larger than one. This is for instance the case when P is the linear interpolation operator in 1, 2 or 3 dimensions. However the exact value depends on the level considered and is typically costly to compute accurately, which leads us to consider the simpler case where we only assume that $\kappa_{\min} \geq 1$, in which case (2.52) can be rewritten, at level i as

$$\lambda_1^{i-1} \geq \frac{\lambda_1^i}{\sigma_i}.$$

Once this lower bound is computed, the algorithm then proceeds to increase λ^L (in a manner that we describe below) if evidence of indefiniteness of $H_r(\lambda)$ is found. We have considered two ways to obtain this evidence. The first is to attempt to solve the system $H_r(\lambda)s = -g$ for the step at level r by a multigrid technique, and to monitor the curvature terms $\langle d, H_i(\lambda)d\rangle$ occurring in the smoothing iterations at each level i. As soon as one of these terms is shown to be negative, we know from Lemma 2.4.1 that the lower bound λ^L must be increased. The second is to use a multilevel eigenvalue solver like the Rayleigh Quotient Minimization Multigrid (RQMG) Algorithm (see (Mandel and McCormick 1989)) to compute λ_1^r, the smallest eigenvalue of H_r, associated with the eigenvector u_1^r. The RQMG algorithm solves the variational problem

$$RQ(u_1^r) = \min_{u\neq 0} RQ(u) = \min_{u\neq 0} \frac{\langle H_r u, u\rangle}{\langle u, u\rangle}$$

by applying a smoothing strategy adapted to the Rayleigh quotient minimization at each level i . The solution to this problem is an (upper) approximation to λ_1^r which, if negative, may therefore be used to deduce the bound $\lambda^L \geq -\lambda_1^r$. Observe that the RQMG algorithm (applied with sufficient accuracy) ensures that $H_r(\lambda^L)$ is, at least in inexact arithmetic, positive semidefinite.

In addition to the lower bound λ^L (which applies to all levels), we compute a initial upper bound λ_i^U for each level i as in the Moré-Sorensen algorithm (observe that no information can be obtained from lower levels about λ_i^U). This therefore provides intervals $[\lambda^L, \lambda_i^U]$ for acceptable λ at each level i.

2.4.1.2 Updating λ in the Positive Definite Case

If $\lambda^L = 0$, $H_r(0)$ is positive definite (in inexact arithmetic) and $\|s(0)\|_2 \leq \Delta$, our problem is solved. If this is not the case, our second task is then to adjust $\lambda \geq \lambda^L$ such that (2.49) holds.

We now describe this adjustment procedure at level i, our final intention being to solve it at level r.

Since we are looking for λ that solves the secular equation, we can apply the Newton method to this end as we did in the Moré-Sorensen algorithm. However, in our case, the Cholesky factor L for $H(\lambda)$ is only available at the lowest level. Fortunately, note that

$$\|w\|^2 = \langle w, w\rangle = \langle L^{-1}s, L^{-1}s\rangle = \langle s, L^{-T}L^{-1}s\rangle = \langle s, (H(\lambda))^{-1}s\rangle.$$

Thus, if we compute y as the solution to the positive definite system

$$H(\lambda)y = s(\lambda), \tag{2.53}$$

the Newton step for the secular equation at the current level then takes the form

$$\lambda^{\text{new}} = \lambda + \left(\frac{\|s\|_i - \Delta}{\Delta}\right)\left(\frac{\|s\|_i^2}{\langle s, y\rangle}\right). \tag{2.54}$$

Since we don't rely on factorizations for an exact solution of the system (2.53), we therefore apply a multigrid method to solve for y. However, this solution may be considered as costly. An alternative option is to update λ by applying a secant method to the secular equation, which gives

$$\lambda^+ = \lambda - \phi(\lambda)\left(\frac{\lambda - \lambda_{\text{old}}}{\phi(\lambda) - \phi(\lambda_{\text{old}})}\right). \tag{2.55}$$

(We use $\lambda_{\text{old}} = \lambda^U$ to start the iteration.)

As in the Moré-Sorensen algorithm, if λ^{new} lies outside the interval, we choose λ inside the interval. One way to do this is to take λ^{new} as the half of the interval $[\lambda^L, \lambda^U]$, which corresponds to a simple bisection step. But we can expect better results by choosing to follow Moré and Sorensen (1979) and setting

$$\lambda^{\text{new}} = \max\left[\sqrt{\lambda^L\lambda^U}, \lambda^L + \psi(\lambda^U - \lambda^L)\right], \tag{2.56}$$

for $\psi \in (0, 1)$, which ensures that λ^{new} is closer to λ^L.

2.4.1.3 The Complete Algorithm

We need to introduce three further comments before the formal statement of the algorithm.

We first note that once a restricted trust-region problem (2.50) has been solved at level i, this means that the corresponding λ can be used as a lower bound for all higher levels. No further updating of λ is therefore necessary at this level and all lower ones, but we may nevertheless continue to exploit level i in the multigrid solution of the linear systems occurring at higher levels. The fact that a solution at level i has already been computed is remembered in our algorithm by setting the flag $\texttt{issolved}_i$. (For coherence, we define these flags for levels $1, \ldots, r+1$.)

Our second comment is that we still need to define stopping criteria for the multigrid solution of (2.50). A first criterion is obviously to terminate the iterations when the residual of the system is sufficiently small. In practice, we choose to stop the solution of the system as soon as

$$\|r_{i,k}\| = \| - g_i - H_i(\lambda)s_{i,k}\| \leq \epsilon^r,$$

with $\epsilon^r \in (0,1)$. However, we might need to introduce a second stopping rule. It may indeed happen that, for a current λ (too small), the step resulting from the system has a i-norm exceeding Δ. It is of course wasteful to iterate too long to discover, upon termination, that we have to throw the solution away. In order to avoid this wasteful calculation, we exploit the fact that the norm of the multigrid iterates is increasing as the iterations proceed. Thus, if this norm exceeds Δ by some threshold D^{++}, we decide to terminate the iterative process (and subsequently increase λ). However, we must be careful not to alter the lower and upper bounds on λ in this subsequent update, because of the possible inaccuracy generated by the early truncation of the system and the absence of any monotonicity guarantee (at variance with methods like truncated conjugate-gradients, see (Steihaug 1983)). Unfortunately, it is also possible that no λ in the current interval produces a sufficiently small step. In this case, λ grows and becomes arbitrarily close to its upper bound. We avoid this situation by increasing our threshold whenever λ is within ϵ_i^λ of λ_i^U.

Finally, we have to propagate changes in λ between levels. Thus, if we have just updated λ and the old one was λ^-, we have that

$$H_i(\lambda) = H_i(\lambda^-) + (\lambda - \lambda^-) M_i Q_i. \tag{2.57}$$

Similarly, taking into account that each residual at level ℓ is computed with respect to the linear system at level $\ell + 1$, we have that the residual is given by

$$r_{i,k+1} = -g_i - \sum_{\ell=i}^{r} M_\ell H_\ell(\lambda) s_{\ell,k}, \tag{2.58}$$

where we have use the fact that the right-hand-side of the system at level i is the restricted residual of level $i+1$. And, by introducing (2.57) in (2.58),

$$\begin{array}{rcl} r_{i,k+1} & = & -g_i \\ & & -\sum_{\ell=i}^{r} M_\ell H_\ell(\lambda^-) s_{\ell,k} \\ & & -\sum_{\ell=i}^{r} M_\ell (\lambda - \lambda^-) M_\ell Q_\ell s_{\ell,k}, \end{array} \tag{2.59}$$

and thus, by grouping terms and by the definition of $r_{i,k}$, the residual update satisfies

$$r_{i,k+1} = r_{i,k} - \sum_{\ell=i}^{r} M_\ell (\lambda - \lambda^-) M_\ell Q_\ell s_{\ell,k}, \tag{2.60}$$

where $s_{\ell,k}$ is the current iterate computed in level ℓ.

We now present the complete multigrid algorithm for the solution of the trust-region subproblem, the Multigrid Moré-Sorensen (MMS) Algorithm 2.4.1 on the following page. Note that for each level i, we start by unsetting $\texttt{issolved}_i$.

Some comments on this algorithm are necessary at this point.

1. The algorithm is called form the virtual level $r+1$, after an initialization phase which computes, once and for all and for every level, the values of $D^+ = (1+\theta)\Delta$, $D^- = (1-$

Algorithm 2.4.1: [$s_{i,*}$,λ_i] = MMS(i, H_i, $r_{i,0}$, Δ, λ^L, λ, $s_{i,0}$, $\texttt{issolved}_i$)

Step 0. Initialization. Set $k = 0$.

Step 1. Iteration Choice. If $i = 1$, go to Step 3. Otherwise, if (2.51) fails, go to Step 4 (Smoothing iteration). Else, choose to go to Step 2 or to Step 4.

Step 2. Recursive Iteration. Call MMS recursively as follows:

$$[e_{i-1,*}, \lambda_{i-1}] = \text{MMS}(i-1, H_{i-1}, r_{i-1,0}, \Delta, \lambda^L, \lambda, 0_{i-1}, \texttt{issolved}_{i-1})$$

where $r_{i-1,0} = R_i r_{i,k}$ and 0_{i-1} is the zero vector of size n_{i-1}. Compute $s_{i,k+1} = s_{i,k} + P_i e_{i-1,*}$. If $\texttt{issolved}_i$ is unset, i.e. this is the first time we perform a recursive iteration at this level, set $\lambda^L = \lambda_{i-1}$, choose $\lambda \in [\lambda^L, \lambda_i^U]$ using (2.56), update $H_i(\lambda)$ using (2.57) and $r_{i,k+1}$ using (2.60) and set $\texttt{issolved}_i$. Go to Step 5.

Step 3. Exact Iteration. If $\texttt{issolved}_{i+1}$ is unset, call the Moré-Sorensen algorithm, returning with solution [$s_{i,*}$,λ_i] = MS($r_{i,0}$,$H_i(\lambda)$,Δ), and set $\texttt{issolved}_i$. Otherwise, just solve the system $H_i(\lambda)s_{i,*} = r_{i,0}$ exactly by Cholesky factorization of $H_i(\lambda)$ and return with solution $(s_{i,*}, \lambda)$.

Step 4. Smoothing Iteration. Apply μ smoothing cycles on the residual equation $H_i(\lambda)e_{i,k} = r_{i,k}$ yielding $s_{i,k+1}$, set $r_{i,k+1} = r_{i,k} + H_i(\lambda)(s_{i,k+1} - s_{i,k})$ and go to Step 5.

Step 5. Termination. If $\|r_{i,k+1}\| < \epsilon^r$ and $\texttt{issolved}_{i+1}$ is set, return $s_{i,k+1}$ and λ. Else, go to Step 1 if $\texttt{issolved}_{i+1}$ is set or if $\|r_{i,k+1}\| \geq \epsilon^r$ and $\|s_{i,k+1}\|_i \leq D^{++}$.

Step 6. Parameter update after full system solution.
If $\|r_{i,k+1}\| < \epsilon^r$ (and $\texttt{issolved}_{i+1}$ is unset),

Step 6.1: step threshold update. If $\lambda_i^U - \lambda < \epsilon_i^\lambda$, set $D^{++} = 2D^{++}$.

Step 6.2: interior solution test. If $\lambda = 0$ and $\|s_{i,k+1}\|_i < D^+$, or if $\lambda \geq 0$ and $D^- \leq \|s_{i,k+1}\|_i \leq D^+$, return with solution $s_{i,*} = s_{i,k+1}$ and $\lambda_i = \lambda$.

Step 6.3: parameter and interval updates. If $\|s_{i,k+1}\|_i > D^+$, set $\lambda^L = \lambda$. If $\|s_{i,k+1}\|_i < D^-$, set $\lambda_i^U = \lambda$. Compute a new $\lambda \in [\lambda^L, \lambda_i^U]$ using (2.54) or (2.55).

Step 6.4: reset the step. Set $s_{i,k+1} = 0$, $r_{i,k+1} = r_{i,0}$, update $H_i(\lambda)$ using (2.57), and go to Step 1.

Step 7: Parameter update after incomplete system solution.
If $\|r_{i,k+1}\| \geq \epsilon^r$ (and $\|s_{i,k+1}\|_i > D^{++}$),

Step 7.1: parameter update. compute a new $\lambda \in [\lambda, \lambda_i^U]$ using (2.54) or (2.55).

Step 7.2: reset the step. Set $s_{i,k+1} = 0$, $r_{i,k+1} = r_{i,0}$, update $H_i(\lambda)$ using (2.57), and go to Step 1.

$\theta)\Delta$ and $D^{++} = \sigma_i D^+$ for some $\theta \in (0, 1)$. A level-dependent feasible interval $[\lambda^L, \lambda_i^U]$ is also computed at this stage. The (global) lower bound λ^L is set to the maximum between 0 and the opposite of the approximation of the most negative eigenvalue produced by the RQMG algorithm; the upper bound is calculated, for each level, exactly as for the Moré-Sorensen algorithm, using the appropriate restrictions of the gradient and Hessian to the considered level. An initial value of $\lambda \in [\lambda^L, \lambda_i^U]$ is finally computed using (2.56) before the call to MMS proper.

2. We may essentially identify Steps 0 to 5 as a classical multigrid solver for a linear system when $\texttt{issolved}_i$ is set. The remaining contain the update to the λ parameter, broadly following the Moré-Sorensen method.

3. The linear system (2.44) is solved by computing a correction at coarse levels to the steps already computed at finer ones. Our restriction strategy produces an algorithm analog to the application, in our nonlinear context, of the Full Multigrid Scheme.

4. We have not specified the details of the smoothing procedure in Step 4. In our experiments, we have used the Gauss-Seidel smoother.

2.4.2 Preliminary Numerical Experience

In this section we present some numerical results obtained by two variants of the MMS method applied in a trust-region algorithm for four different test problems involving three-dimensional discretizations. Some of these problems were also tested in Gratton, Sartenaer and Toint (2006) in their two-dimensional formulation. All problems presented here are defined on the unit three-dimensional cube S_3 and tested with a fine discretization of 63^3 variables, and we used 4 levels of discretization. The Laplacian operator is obtained from the classical 7-points pencil. The prolongation operator is given by linear interpolation, and the restriction as its normalized (in the $\|\cdot\|_1$ norm) transpose, thereby defining $\sigma_i = \|P_i\|_1$. We briefly review these test problems below.

2.4.2.1 Optimization Test Problems

3D Quadratic Problem 1 (3D-1): A convex quadratic problem, where we consider the three-dimensional boundary value problem defined by

$$\begin{aligned} -\Delta u(x,y,z) &= f \quad \text{in } S_3 \\ u(x,y,z) &= 0 \quad \text{on } \partial S_3, \end{aligned}$$

where f is chosen so that the analytical solution to this problem is $u(x, y, z) = 8$. This gives linear systems $A_i x = b_i$ at level i where each A_i is a symmetric positive definite matrix. This problem is the typical *model problem* for multigrid solvers. Here, we want to find the solution to its variational formulation

$$\min_{x \in \mathbb{R}^{n_r}} \frac{1}{2} x^T A_r x - x^T b_r.$$

3D Nonlinear Problem 2 (3D-2): Another convex quadratic problem, where we consider the differential equation

$$\begin{aligned} -(1+\sin(3\pi x)^2)\Delta u(x,y,z) &= f \quad \text{in } S_3 \\ u(x,y,z) &= 0 \quad \text{on } \partial S_3, \end{aligned}$$

where f is chosen so that the analytical solution to this problem is

$$u(x,y,z) = x(1-x)y(1-y)z(1-z).$$

This problem is again considered in its variational formulation, as for problem 3D-1.

Boundary Value Problem (BV): This is a problem inspired by the one dimensional two-point boundary value problem presented in Moré, Garbow and Hillstrom (1981) and is defined by

$$-\Delta u(s,t,z) = \tfrac{1}{2}(u(s,t,z) + t + s + z + 1)^3,$$

with

$$\begin{aligned} u(0,t,z) = u(1,t,z) = 0, \quad 0 < t < 1, \\ u(s,0,z) = u(s,1,z) = 0, \quad 0 < s < 1, \\ u(s,t,0) = u(s,t,1) = 0, \quad 0 < z < 1. \end{aligned}$$

Here, we look for the solution of the least squares problem

$$\min_{s,t,z \in [0,1]} \| -\Delta u(s,t,z) - \tfrac{1}{2}(u(s,t,z) + t + s + z + 1)^3 \|_2^2.$$

3D Poisson Problem (Poisson): Here we consider the variational formulation of the problem defined by the nonlinear partial differential equation

$$\Delta u = \frac{u^3}{1 + x^2 + y^2 + z^2},$$

with Dirichlet boundary conditions $u = 0$ on ∂S_3.

Bratu problem (Bratu): We want to solve the variational formulation of the following nonlinear partial differential equation

$$\Delta u + R\exp(u) = 0, \qquad R = 6.8,$$

over S_3 with Dirichlet boundary conditions $u = 0$ on ∂S_3.

2.4.2.2 Numerical Results

We discuss here results obtained by applying the simple BTR trust-region method for the minimization of these problems, in which the subproblem is solved[3] at each iteration by one of three multigrid variants of the Moré-Sorensen algorithm. The first variant (MMS-secant)

[3] We require the Euclidean norm gradient of the objective function to be at most 10^{-6} for termination.

is the MMS algorithm where we use the secant approach (2.55) to solve the secular equation. The second (MMS-Newton) is the same method, but using Newton's method (2.54) instead of (2.55). The third (naive MMS-secant) is a simpler version of MMS-secant in which we do not use information on λ from lower levels. In this variant, we solve the Moré-Sorensen system (2.44) by multigrid instead of using Cholesky factorization of the Hessian, but we only change λ at the topmost level. This is equivalent to setting $\texttt{issolved}_i$ for all levels $i < p + 1$. We update λ by using the secant method on the secular equation, as described above. All runs were performed in Matlab v.7.1.0.183 (R14) Service Pack 3 on a 3.2 GHz Intel single-core processor computer with 2 Gbytes of RAM, using the parameters

$$\mu = 5, \quad \theta = 0.1, \quad \epsilon^r = 10^{-6}, \quad \psi = 10^{-4}, \text{ and } \epsilon_i^\lambda = 0.01|\lambda_i^U - \lambda^L|.$$

Our results are shown in Table 2.1. In this table, $\#\lambda$ stands for the weighted number of λ-updates, where each update is weighted proportionally to the dimension of the subspace in which the update is performed. Similarly, $\#sys$ stands for the the weighted number of linear systems solved, and $\#R$ for the weighted number of restrictions performed by the algorithm. This last number indicates how many recursive iterations were used to find the solution of the linear system over the course of optimization.

	Naive MMS-secant			MMS-secant			MMS-Newton		
	$\#\lambda$	$\#sys$	$\#R$	$\#\lambda$	$\#sys$	$\#R$	$\#\lambda$	$\#sys$	$\#R$
3D-1	54	54	184.14	48.18	48.18	191.76	41.75	83.50	179.07
3D-2	63	63	204.56	42.44	42.44	165.92	59.56	119.12	218.42
BV	249	249	145.81	149.07	149.07	81.34	198.17	396.34	115.10
Poisson	43	43	25.87	25.41	25.41	19.07	20.51	41.02	29.90
Bratu	27	27	20.25	21.28	21.28	15.68	23.0	46.00	36.65

Table 2.1: Results for three variants of the MMS method (with RQMG).

Because the global efficiency is dominated by the number of linear solves $\#sys$, these results demonstrate that the best approach is the MMS-secant. Moreover, this variant also outperforms the naive version in terms of the number of λ-updates required to solve all the trust-region subproblems in an optimization run. We also note that MMS-Newton does not offer a significant advantage over MMS-secant. Even when less λ-updates are needed to find the solution, these updates are computationally much more expensive than the simple secant ones since an additional linear system must be solved by multigrid for each update, which results in a slower algorithm. The proportionally larger numbers of updates, system solves and restrictions for problem BV are caused by its nonconvexity. The results of Table 2.1 therefore indicate that information obtained at lower levels is useful and should therefore be exploited.

In all of these results, we computed an initial λ^L using an estimate of the smallest eigenvalue of the Hessian in each BTR iteration by means of the RQMG algorithm. We note that, for the convex problems, RQMG does not give us a better estimate than the initial zero λ^L, given that the opposite of the smallest eigenvalue is still negative. As a consequence, this options seems unnecessary if the problem is known to be convex. Indeed, the cost of obtaining our initial

estimate by RQMG is equivalent to the solution of one linear system by multigrid, and can be substantial. While it corresponds to approximately 5% of the total time required to solve the optimization problem for 3D-1 and 3D-2, 15% for Poisson and Bratu, this cost can be as high as 50% for the nonconvex problem BV. This price is paid for a strong guarantee of convergence to minimizers. It can be reduced by only using this option in the neighborhood of a first-order critical point [(4)], or even completely dismissed if we are ready to accept weaker guarantees on the limit points produced by the algorithm (indeed, Gratton et al. (2008*b*) have shown that the 2-norm RMTR algorithm converges to *weakly second-order critical points*, that is points that are second-order critical in a subspace defined by the directions generated by the points of the discretization of the problem).

We have developed a new method for the exact solution of the trust-region subproblem which is suitable for large scale systems where the Moré-Sorensen method cannot be applied, for instance because factorizations are too costly or impossible. This method exploits the multigrid structure in order to extract curvature information from the coarse levels to speed up the computation of the Lagrange parameter associated with the subproblem.

We have presented some admittedly limited numerical experience, which shows the potential for the new method, both because it demonstrates that sizable three-dimensional applications can be considered and because it outperforms a too naive multigrid implementation of the basic Moré-Sorensen algorithm. However, additional numerical results have shown that the method is outperformed by approximate solvers like the truncated conjugate gradient method. This is why, we only consider approximate solvers in the following. We present in the next section the modification by the multigrid principles of the trust-region algorithm.

2.5 The recursive multilevel trust-region algorithm

2.5.1 Description of the method

Now, we consider the bound-constrained optimization problem

$$\min_{x \in \mathcal{F}} f(x), \tag{2.61}$$

where f is a twice-continuously differentiable objective function which maps $\mathbb{R}^n$ into $\mathbb{R}$ and is bounded below, where $\mathcal{F} = \{x \in \mathbb{R}^n \,|\, l \leq x \leq u\}$ is a set of bound constraints and where $l, u \in \mathbb{R}^n$ and are possibly infinite.

Many practical trust-region algorithms, including that presented here, use a *quadratic model*

$$m_k(x_k + s) = f(x_k) + \langle g_k, s\rangle + \tfrac{1}{2}\langle s, H_k s\rangle, \tag{2.62}$$

where $g_k \stackrel{\text{def}}{=} \nabla f(x_k)$, H_k is a symmetric $n \times n$ approximation of $\nabla^2 f(x_k)$, and $\langle \cdot, \cdot \rangle$ is the Euclidean inner product. A sufficient decrease in this model inside the trust region is then

[(4)]For problem BV, applying the RQMG option only when the $\|g\|$ is below 10 times the stopping threshold reduces the additional cost from 50% to 10%.

obtained by (approximately) solving

$$\begin{array}{ll} \min & m_k(x_k + s). \\ \|s\|_\infty \leq \Delta_k & \\ x_k + s \in \mathcal{F} & \end{array} \tag{2.63}$$

The choice of the infinity norm in the trust-region description is natural in the context of bound-constrained problems, because the feasible set for problem (2.63) can then be fully represented by bound constraints.

As proposed in Gratton et al. (2008*b*) and further explored in Gratton et al. (2008*a*), we consider exploiting the knowledge of a hierarchy of descriptions for problem (2.61), if such a hierarchy is known. To be more specific, suppose that a collection of functions $\{f_i\}_{i=0}^r$ is available, each f_i being a twice-continuously differentiable function from $\mathbb{R}^{n_i}$ to $\mathbb{R}$ (with $n_i \geq n_{i-1}$). We assume that $n_r = n$ and $f_r(x) = f(x)$ for all $x \in \mathbb{R}^n$, giving back our original problem. We also make the assumption that f_i is "more costly" to minimize than f_{i-1} for each $i = 1, \ldots, r$. This is typically the case if the f_i represent increasingly finer discretizations of the same infinite-dimensional objective. To fix terminology, we will refer to a particular i as a *level*. We use the first subscript i in all subsequent subscripted symbols to denote a quantity corresponding to the i-th level, ranging from coarsest ($i = 0$) to finest ($i = r$) (meaning in particular, if applied to a vector, that this vector belongs to $\mathbb{R}^{n_i}$). Some relation must exist between the variables of two successive functions of the collection set $\{f_i\}_{i=0}^r$. We thus assume that, for each $i = 1, \ldots, r$, there exist a full-rank linear operator R_i from $\mathbb{R}^{n_i}$ into $\mathbb{R}^{n_{i-1}}$ (the restriction) and another full-rank linear operator P_i from $\mathbb{R}^{n_{i-1}}$ into $\mathbb{R}^{n_i}$ (the prolongation) such that

$$\sigma_i P_i = R_i^T, \tag{2.64}$$

for some known constant $\sigma_i > 0$, where P_i and R_i are interpreted as restriction and prolongation between a fine and a coarse grid. We assume that the restriction operators are normalized to ensure that $\|R_i\|_\infty = 1$ and also that the entries of R_i and P_i are all non-negative.

The philosophy of our recursive algorithm is then to use the hierarchy of problem descriptions $\{f_i\}_{i=0}^{r-1}$ to efficiently construct minimizations steps. More precisely, we build, for each level i, a model leading to a *local bound-constrained minimization subproblem* at the coarse level $i-1$, and then compute a coarse step by *solving this subproblem using a trust-region algorithm*. The resulting coarse move is then prolongated (using P_i) into a trust-region step at level i. For this purpose, we first need to build an alternative *local lower-level model* h_{i-1} representing at level $i-1$ the function h_i to be minimized at level i (with $h_r = f_r = f$). We also need to define a set of bound constraints which represents both the *feasibility* with respect to the original (finest level) bounds and the *constraints on the stepsize* inherited from the trust regions at level i as well as at levels $i+1, \ldots, r$.

Consider first the construction of the local lower-level model h_{i-1} of h_i around $x_{i,k}$, the iterate at some iteration k at level i, say. If we restrict $x_{i,k}$ to level $i-1$ and define $x_{i-1,0} = R_i x_{i,k}$, the model h_{i-1} is then given by

$$h_{i-1}(x_{i-1,0} + s_{i-1}) \stackrel{\text{def}}{=} f_{i-1}(x_{i-1,0} + s_{i-1}) + \langle v_{i-1}, s_{i-1} \rangle, \tag{2.65}$$

where $v_{i-1} = R_i g_{i,k} - \nabla f_{i-1}(x_{i-1,0})$ with $g_{i,k} \stackrel{\rm def}{=} \nabla h_i(x_{i,k})$. By convention, we set $v_r = 0$, such that, for all s_r,

$$h_r(x_{r,0} + s_r) = f_r(x_{r,0} + s_r) = f(x_{r,0} + s_r) \;\text{ and }\; g_{r,k} = \nabla h_r(x_{r,k}) = \nabla f(x_{r,k}).$$

The model h_{i-1} thus results from a *modification* of f_{i-1} by a linear term that enforces the relation $g_{i-1,0} = \nabla h_{i-1}(x_{i-1,0}) = R_i g_{i,k}$. This first-order modification[(5)] ensures that the first-order behaviors of h_i and h_{i-1} are similar in a neighborhood of $x_{i,k}$ and $x_{i-1,0}$, respectively. Indeed, if s_i and s_{i-1} satisfy $s_i = P_i s_{i-1}$, we then have that

$$\langle g_{i,k}, s_i \rangle = \langle g_{i,k}, P_i s_{i-1} \rangle = \frac{1}{\sigma_i} \langle R_i g_{i,k}, s_{i-1} \rangle = \frac{1}{\sigma_i} \langle g_{i-1,0}, s_{i-1} \rangle, \tag{2.66}$$

where we have also used (2.64).

We next need to represent, at level $i-1$, feasibility with respect to the bound constraints. Because we aim at a description which is coherent across levels and because we wish to avoid general linear constraints, we choose this representation as a bound-constrained domain $\mathcal{F}_{i-1}$ defined recursively such that, for $i = 1, \ldots, r$,

$$x_{i,k} + P_i s_{i-1} \in \mathcal{F}_i \text{ for all } x_{i-1,0} + s_{i-1} \in \mathcal{F}_{i-1}, \tag{2.67}$$

with $\mathcal{F}_r = \mathcal{F}$, thereby ensuring that all iterates at the finest level remain feasible for the original bounds. In our algorithm, the specific choice of $\mathcal{F}_{i-1}$ is done using a Gelman-Mandel-like formula (see Gelman and Mandel, 1990, or Gratton et al., 2008*a*), stating that

$$\mathcal{F}_{i-1} = \{x_{i-1} \in \mathbb{R}^{n_{i-1}} \;\mid\; l_{i-1} \leq x_{i-1} \leq u_{i-1}\}, \tag{2.68}$$

where the bound vectors l_{i-1} and u_{i-1} are recursively defined componentwise by

$$[l_{i-1}]_j = [x_{i-1,0}]_j + \frac{1}{\|P_i\|_\infty} \max_{t=1,\ldots,n_i} [l_i - x_{i,k}]_t \tag{2.69}$$

and

$$[u_{i-1}]_j = [x_{i-1,0}]_j + \frac{1}{\|P_i\|_\infty} \min_{t=1,\ldots,n_i} [u_i - x_{i,k}]_t, \tag{2.70}$$

for $j = 1, \ldots, n_{i-1}$, with $l_r = l$ and $u_r = u$. We refer the reader to Lemma 4.3 in Gratton et al. (2008*a*) for a proof of (2.67).

We then need to represent at the coarser level $i-1$ the constraints on the stepsize resulting from the trust region at level i,

$$\mathcal{B}_{i,k} = \{x_{i,k} + s_i \in \mathbb{R}^{n_i} \;\mid\; \|s_i\|_\infty \leq \Delta_{i,k}\},$$

associated with $x_{i,k}$, and also from the trust regions at levels higher than i. Let us denote by $\mathcal{A}_i$ the box representing these stepsize constraints inherited from higher levels. Then $\mathcal{B}_{i,k} \cap \mathcal{A}_i$ is

[(5)] The first-order modification (2.65) is usual in multigrid applications in the context of the "full approximation scheme", where it is usually called the "tau correction" (see, for instance, Chapter 3 of Briggs et al., 2000, or Hemker and Johnson, 1987).

also a box of the form $\{x_i \mid v_i \leq x_i \leq w_i\}$, where $v_i, w_i \in \mathbb{R}^{n_i}$. The set $\mathcal{A}_{i-1}$ is then defined by

$$\mathcal{A}_{i-1} = \{x_{i-1} \in \mathbb{R}^{n_{i-1}} \mid R_i v_i \leq x_{i-1} \leq R_i w_i\}. \tag{2.71}$$

For consistency, we set $\mathcal{A}_r = \mathbb{R}^n$. This definition is less restrictive than the Gelman-Mandel procedure used to define $\mathcal{F}_{i-1}$, but does not imply that $x_{i,k} + P_i s_{i-1} \in \mathcal{B}_{i,k} \cap \mathcal{A}_i$ for all $x_{i-1,0} + s_{i-1} \in \mathcal{A}_{i-1}$. This remains acceptable because the trust-region bounds need only be satisfied up to a constant factor to ensure global convergence (again see Gratton et al., 2008*a*).

At level $i-1$, we finally consider the intersection of the domain corresponding to the original bounds on the problem and that resulting from the trust-region restrictions at higher level (if any). This intersection is given by $\mathcal{L}_i \stackrel{\text{def}}{=} \mathcal{F}_i \cap \mathcal{A}_i$ for $i = 0, \ldots, r$. The local subproblem to be solved at level $i-1$ is then given by

$$\min_{x_{i-1,0} + s_{i-1} \in \mathcal{L}_{i-1}} h_{i-1}(x_{i-1,0} + s_{i-1}).$$

As indicated above, we solve this subproblem using a trust-region method starting from $x_{i-1,0}$, whose ℓ-th iteration then involves the computation of

$$\min_{\substack{\|s_{i-1}\|_\infty \leq \Delta_{i-1,\ell} \\ x_{i-1,\ell} + s_{i-1} \in \mathcal{L}_{i-1}}} h_{i-1}(x_{i-1,\ell} + s_{i-1}). \tag{2.72}$$

In addition to the features already discussed, our recursive multilevel trust-region algorithm (see Algorithm RMTR$_\infty$ on page 77) crucially considers whether recurring to the local lower-level model is useful. This decision is made by comparing *criticality measures* at the current and lower levels. At level i, the criticality measure is computed as

$$\chi_{i,k} \stackrel{\text{def}}{=} \chi(x_{i,k}) = |\min_{\substack{x_{i,k} + d \in \mathcal{L}_i \\ \|d\|_\infty \leq 1}} \langle g_{i,k}, d\rangle|, \tag{2.73}$$

which can be interpreted as the maximal decrease of the linearized problem that can be achieved in the intersection of $\mathcal{L}_i$ and a box of radius one (see Conn et al., 1993, for instance). We declare that recurring to the lower level is useful whenever this decrease at level $i-1$ is significant compared to that achievable at level i, which we formalize by the condition that

$$\frac{\chi_{i-1,0}}{\sigma_i} \geq \kappa_\chi \chi_{i,k}, \tag{2.74}$$

where $\kappa_\chi \in (0, 1)$. The factor σ_i in the left-hand side results from (2.66) and the fact that the criticality measure (2.73) is a linear approximation of the decrease that can be achieved from $x_{i,k}$. If (2.74) does not hold, then the algorithm resorts to using the quadratic model (2.62) at level i, which we denote by $m_{i,k}(x_{i,k} + s_i)$. Otherwise, the choice between the two models remains open, allowing, as we discuss below, the efficient exploitation of multigrid techniques such as *smoothing iterations*.

We now turn to the more formal description of our algorithm and assume that the prolongations P_i and the restrictions R_i are known, as well as the functions $\{f_i\}_{i=0}^{r-1}$. We use the

constants κ_χ, η_1, η_2, γ_1 and γ_2 satisfying the conditions $\kappa_\chi \in (0,1)$, $0 < \eta_1 \leq \eta_2 < 1$, and $0 < \gamma_1 \leq \gamma_2 < 1$. An initial trust-region radius for each level, $\Delta_{i,0} > 0$, is also defined. The algorithm's initial data consists of the level index i ($0 \leq i \leq r$), a starting point $x_{i,0}$, the gradient $g_{i,0}$ at this point and the corresponding criticality measure $\chi_{i,0}$, the description of the feasible sets $\mathcal{F}_i$ and $\mathcal{A}_i$, and a criticality tolerance $\epsilon_i^\chi \in (0,1)$.

Further motivation for this algorithm can be found in Gratton et al. (2008*a*), together with a proof that, under reasonable assumptions, every limit point of the sequence of produced iterates must be a first-order critical point in the sense that $\lim_{k\to\infty} \chi_{r,k} = 0$. In particular, the functions f_i must have uniformly bounded Hessians for $i = 0, \ldots, r$. We produce a few additional useful comments :

1. The minimization of $f(x) = f_r(x_r) = h_r(x_r)$ (up to the critical tolerance $\epsilon_r^\chi < \chi_{r,0}$) is achieved by calling RMTR$_\infty$(r, $x_{r,0}$, $g_{r,0}$, $\chi_{r,0}$, $\mathcal{F}$, $\mathbb{R}^n$, ϵ_r^χ), for some starting point $x_{r,0}$. For coherence of notations, we thus view this call as being made from some (virtual) iteration 0 at level $r+1$.

2. The test for the value of i at the beginning of Step 1 is designed to identify the lowest level, at which no further recursion is possible. In this case, a Taylor's (i.e., non-recursive) iteration is the only possibility.

3. The set $\mathcal{W}_{i,k}$ represents the feasible domain of subproblem (2.72).

4. The formula for $\delta_{i,k}$ in Step 2 results from (2.65) and (2.66).

5. The "sufficient decrease" in the model (2.75) imposed in Step 3 means, as usual for trust-region methods, that the step $s_{i,k}$ must satisfy the *Cauchy point condition* (see Chapter 12 of Conn et al., 2000) which imposes sufficient decrease relative to the local first-order behavior of the objective function. It requires that

$$m_{i,k}(x_{i,k}) - m_{i,k}(x_{i,k} + s_{i,k}) \geq \kappa_{\text{red}} \chi_{i,k} \min\left[\frac{\chi_{i,k}}{1 + \|H_{i,k}\|_\infty}, \Delta_{i,k}, 1\right] \tag{2.78}$$

   for some constant $\kappa_{\text{red}} \in (0,1)$.

6. Iteration k at level i is said to be *successful* if $\rho_{i,k} \geq \eta_1$.

2.5.2 Note on the convergence theory

We now briefly cite three important results of the convergence theory (the interested reader may see Gratton et al. (2008*a*) for the complete proof of convergence).

Let us denote (i,k), the k-th iteration at level i. If a recursive step is used at iteration (i,k), we say that this iteration initiates a *minimization sequence* at level $i-1$, which consists of all successive iterations at this level (starting from the point $x_{i-1,0} = R_i x_{i,k}$) until a return is made to level i within iteration (i,k). In this case, we say that iteration (i,k) is the predecessor of the minimization sequence at level $i-1$.

Algorithm 2.5.1: RMTR$_\infty$ $(i, x_{i,0}, g_{i,0}, \chi_{i,0}, \mathcal{F}_i, \mathcal{A}_i, \epsilon_i^\chi)$

Step 0: Initialization. Compute $f_i(x_{i,0})$. Set $k = 0$ and

$$\mathcal{L}_i = \mathcal{F}_i \cap \mathcal{A}_i \quad \text{and} \quad \mathcal{W}_{i,0} = \mathcal{L}_i \cap \mathcal{B}_{i,0},$$

where $\mathcal{B}_{i,0} = \{x_{i,0} + s_i \in \mathbb{R}^{n_i} \mid \|s_i\|_\infty \leq \Delta_{i,0}\}$.

Step 1: Model choice. If $i = 0$, go to Step 3. Else, compute $R_i x_{i,k}$, $R_i g_{i,k}$, $\mathcal{F}_{i-1}$ from (2.68)-(2.70), $\mathcal{A}_{i-1}$ from (2.71) and $\chi_{i-1,0}$. If (2.74) fails, go to Step 3. Otherwise, choose to go to Step 2 or to Step 3.

Step 2: Recursive step computation. Call Algorithm

$$\mathbf{RMTR}_\infty(i-1, R_i x_{i,k}, R_i g_{i,k}, \chi_{i-1,0}, \mathcal{F}_{i-1}, \mathcal{A}_{i-1}, \epsilon_{i-1}^\chi),$$

yielding an approximate solution $x_{i-1,*}$ of (2.72). Then define $s_{i,k} = P_i(x_{i-1,*} - R_i x_{i,k})$, set $\delta_{i,k} = \frac{1}{\sigma_i}\big[h_{i-1}(R_i x_{i,k}) - h_{i-1}(x_{i-1,*})\big]$ and go to Step 4.

Step 3: Taylor step computation. Choose $H_{i,k}$ and compute a step $s_{i,k} \in \mathbb{R}^{n_i}$ that sufficiently reduces the model

$$m_{i,k}(x_{i,k} + s_i) = h_i(x_{i,k}) + \langle g_{i,k}, s_i\rangle + \tfrac{1}{2}\langle s_i, H_{i,k} s_i\rangle \tag{2.75}$$

and such that $x_{i,k} + s_{i,k} \in \mathcal{W}_{i,k}$. Set $\delta_{i,k} = m_{i,k}(x_{i,k}) - m_{i,k}(x_{i,k} + s_{i,k})$.

Step 4: Acceptance of the trial point. Compute $h_i(x_{i,k} + s_{i,k})$ and

$$\rho_{i,k} = \big[h_i(x_{i,k}) - h_i(x_{i,k} + s_{i,k})\big]/\delta_{i,k}. \tag{2.76}$$

If $\rho_{i,k} \geq \eta_1$, then define $x_{i,k+1} = x_{i,k} + s_{i,k}$; otherwise, define $x_{i,k+1} = x_{i,k}$.

Step 5: Termination. Compute $g_{i,k+1}$ and $\chi_{i,k+1}$. If $\chi_{i,k+1} \leq \epsilon_i^\chi$ or $x_{i,k+1} \notin \mathcal{A}_i$, then return with the approximate solution $x_{i,*} = x_{i,k+1}$.

Step 6: Trust-Region Update. Set

$$\Delta_{i,k+1} \in \begin{cases} [\Delta_{i,k}, +\infty) & \text{if } \rho_{i,k} \geq \eta_2, \\ [\gamma_2 \Delta_{i,k}, \Delta_{i,k}] & \text{if } \rho_{i,k} \in [\eta_1, \eta_2), \\ [\gamma_1 \Delta_{i,k}, \gamma_2 \Delta_{i,k}] & \text{if } \rho_{i,k} < \eta_1, \end{cases} \tag{2.77}$$

and $\mathcal{W}_{i,k+1} = \mathcal{L}_i \cap \mathcal{B}_{i,k+1}$ where

$$\mathcal{B}_{i,k+1} = \{x_{i,k+1} + s_i \in \mathbb{R}^{n_i} \mid \|s_i\|_\infty \leq \Delta_{i,k+1}\}.$$

Increment k by one and go to Step 1.

The first result states that each minimization sequence, that is a recursive call of the algorithm, contains at least one successful iteration. This result is quite important since it allows us to use the multigrid schemes as V-cycles and W-cycles, which are known to be efficient in the case of the solution of linear systems.

Theorem 2.5.1 Each minimization sequence contains at least one successful iteration.

The second result states that the number of iterations in each level is finite which is of course important. This means that, if we call the algorithm recursively with a given tolerance, the algorithm finishes in a finite number of iterations. A bound on the maximum number of iterations may be proved but is not presented here. The result also provides a lower bound on the decrease obtained on the function after a given number of iterations.

Theorem 2.5.2 The number of iterations in each level is finite. Moreover, there exists $\kappa_{\text{h}} \in (0, 1)$ such that, for every minimization sequence at level $i = 0, \ldots, r$ and every $t \geq 0$

$$f_i(x_{i,0}) - f_i(x_{i,t+1}) \geq \tau_{i,t} \mu^{i+1} \kappa_{\text{h}} \tag{2.79}$$

where $\tau_{i,t}$ is the total number of successful Taylor iterations before iteration t at all the coarser levels and the current level and $\mu = \frac{\eta_1}{\sigma_{\max}}$ with $\sigma_{\max} = \max\{1, \max_{i=1\ldots,r} \sigma_i\}$.

And finally, the last result states that if the algorithm is called successively with a sequence of tolerances converging to zero, then the algorithm converges to first-order critical points.

Theorem 2.5.3 Assume that ϵ_r^{χ} is "driven to zero" in Algorithm RMTR$_\infty$. Then

$$\lim_{k \Rightarrow \infty} \chi_{r,k} = 0. \tag{2.80}$$

Chapter 3

RMTR, a Fortran package on multilevel optimization

3.1 Towards a practical algorithm

The description of the RMTR algorithm so far leaves a number of practical choices unspecified. It is the purpose of this section to provide the missing details for our particular implementation. These details are of course influenced by our focus on discretized problems, where the different levels correspond to different discretization grids, from coarser to finer although we try to remain as general as possible.

3.1.1 Taylor iterations: smoothing and solving

The most important issue is how to enforce sufficient decrease at Taylor iterations, that is, when Step 3 of Algorithm 2.5.1 is executed. At the coarsest level ($i = 0$), the cost of fully minimizing the Taylor model (2.75) inside the trust region remains small, since the subproblem is of low dimension. We thus solve the subproblem using the PTCG (Projected Truncated Conjugate-Gradient) algorithm designed for the standard trust-region algorithm described in Algorithm 1.4.5.

At finer levels ($i > 0$), we use an adaptation of multigrid smoothing techniques to the computation of a Taylor step satisfying the requirements of Step 3 of Algorithm RMTR$_\infty$. A very well-known multigrid smoothing technique is the Gauss-Seidel method, in which each equation of the Newton system is solved in succession. To extend this procedure to our case, rather than successively solving equations, we perform successive one-dimensional bound-constrained minimizations of the model (2.75) along the coordinate axes. More precisely, consider the minimization of (2.75) at level i along the j-th axis (starting each minimization from s such that $\nabla m_{i,k}(x_{i,k} + s) \stackrel{\rm def}{=} g$). Then, provided that the j-th diagonal entry of $H_{i,k}$ is positive, the j-th one-dimensional minimization then results in the updates

$$\alpha_j = \Pr_{\mathcal{W}_{i,k}}(-[g]_j/[H_{i,k}]_{jj}), \quad [s]_j \leftarrow [s]_j + \alpha_j \ \text{ and } \ g \leftarrow g + \alpha_j H_{i,k} e_{i,j}, \tag{3.1}$$

where $\Pr_{\mathcal{W}_{i,k}}(.)$ is the orthogonal projection on the intersection of all the constraints at level i, that is on $\mathcal{W}_{i,k} = \mathcal{F}_i \cap \mathcal{A}_i \cap \mathcal{B}_{i,k}$, where we denote by $[v]_j$ the j-th component of the vector v

and by $[M]_{jj}$ the j-th diagonal entry of the matrix M, and where $e_{i,j}$ is the j-th vector of the canonical basis of $\mathbb{R}^{n_i}$. If, on the other hand, $[H_{i,k}]_{jj} \leq 0$, then a descent step is made along the j-th coordinate axis until the boundary of $\mathcal{W}_{i,k}$ is reached and the model gradient is updated accordingly. This process is the well-known Sequential Coordinate Minimization (SCM) (see, for instance, Ortega and Rheinboldt (1970), Section 14.6), which we adapted to handle bound constraints. In what follows, we refer to a set of n_i successive unidimensional minimizations as a *smoothing cycle*. A SCM *smoothing iteration* then consists of one or more of these cycles. This strategy is summarized in Algorithm 3.1.1

Algorithm 3.1.1: Sequential Coordinate Minimization s**=SCM**$(\mathcal{W}_{i,k}, g_{i,k}, H_{i,k})$

Step 0: Initialization. Set $s_{i,k} = 0$ and $x_{i,k+1} = 0$. For each direction $j = 0, \ldots, n$, do

Step 1: Computation of the step. If $[H_{i,k}]_{jj} > 0$, compute $[s^+_{i,k}]_j = \frac{[-g_{i,k}]_j}{[H_{i,k}]_{jj}}$. Otherwise, set

$$[s^+_{i,k}]_j = \begin{cases} [l(\mathcal{W}_{i,k}) - x_k]_j & \text{if } [g_{i,k}]_j > 0, \\ [u(\mathcal{W}_{i,k}) - x_k]_j & \text{if } [g_{i,k}]_j < 0, \\ 0 & \text{if } [g_{i,k}]_j = 0, \end{cases} \tag{3.2}$$

where $l(\mathcal{W}_{i,k})$ and $u(\mathcal{W}_{i,k})$ are the lower and upper bounds of $\mathcal{W}_{i,k}$.

Step 2: Projection on the feasible set. Set $[x_{i,k+1}]_j = Proj_{[\mathcal{W}_{i,k}]_j}([x_{i,k}]_j + [s^+_{i,k}]_j)$ and $[s_{i,k}]_j = [x_{i,k+1}]_j - [x_{i,k}]_j$, where $Proj_{[\mathcal{W}_{i,k}]_j}$ is the orthogonal projection on the feasible set in the j-th direction.

Step 3: Gradient update. For all directions l, set $[g_{i,k}]_l = [g_{i,k}]_l + [H_{i,k}]_{lj}[s_{i,k}]_j$.

In order to enforce convergence to first-order points, we still have to ensure that a sufficient model decrease (2.78) has been obtained within the trust region after one or more complete smoothing cycles. To do so, we start the first smoothing cycle by selecting the axis corresponding to the index j_m such that

$$j_m = \operatorname*{argmin}_j \; [g_{i,k}]_j [d_{i,k}]_j, \tag{3.3}$$

where

$$d_{i,k} = \operatorname*{argmin}_{\substack{x_{i,k} + d \in \mathcal{L}_i \\ \|d\|_\infty \leq 1}} \langle g_{i,k}, d \rangle. \tag{3.4}$$

Indeed in this case the minimization of the model $m_{i,k}$ along $[d_{i,k}]_{j_m}$ within the trust region is guaranteed to yield a *Generalized Cauchy step* $\alpha_{j_m}[d_{i,k}]_{j_m}$ such that the sufficient decrease condition (2.78) holds (as is shown in Lemma 3.1.1). Since the remaining minimizations in

the first smoothing cycle (and the following ones, if any) only decrease the value of the model further, (2.78) thus also holds for the complete step $s_{i,k}$.

Lemma 3.1.1 *Assume that the first unidimensional minimization in the first smoothing cycle at iteration (i,k) is performed along the j_m-th coordinate axis, where j_m is determined by (3.3) and (3.4), and results in a stepsize α_{j_m}. Then (2.78) holds for $s_{i,k} = \alpha_{j_m} e_{i,j_m}$.*

Proof. We drop the indexes i and k for simplicity. First note that

$$|g_{j_m}| \geq |g_{j_m} d_{j_m}| = |\min_j g_j d_j| \geq \frac{1}{n} \left| \sum_j g_j d_j \right| = \frac{1}{n} \chi, \tag{3.5}$$

where we have used (3.4) to derive the first inequality, (3.3) to derive the first equality and the fact that (3.4) implies that $g_j d_j \leq 0$ for all j to deduce the second inequality. Next observe that the line minimization along the j_m-th coordinate axis may terminate in three different situations. The first is when the minimum of the quadratic model is interior to $\mathcal{W}$, in which case we obtain that $H_{j_m,j_m} > 0$, that $\alpha_{j_m} = |g_{j_m}|/H_{j_m,j_m}$ and also that

$$m(x) - m(x + \alpha_{j_m} e_{j_m}) = \frac{|g_{j_m}|^2}{2H_{j_m,j_m}}.$$

Using now the bound $H_{j_m,j_m} \leq 1 + \|H\|_\infty$ and (3.5), we deduce that

$$m(x) - m(x + \alpha_{j_m} e_{j_m}) \geq \frac{\chi^2}{2n^2(1 + \|H\|_\infty)}. \tag{3.6}$$

The second situation is when the line minimizer is on the boundary of $\mathcal{B}$, in which case $\alpha_{j_m} = \Delta$ and thus

$$m(x) - m(x + \alpha_{j_m} e_{j_m}) \geq \frac{1}{2} |g_{j_m}| \Delta \geq \frac{1}{2n} \chi \Delta, \tag{3.7}$$

where we used (3.5) to obtain the last inequality. The third possibility is when the line minimizer is on the boundary of $\mathcal{L}$. In this case, we have that $|\alpha_{j_m}| \geq |d_{j_m}|$, where d is given by (3.4), and therefore, using (3.5) again, that

$$m(x) - m(x + \alpha_{j_m} e_{j_m}) \geq \frac{1}{2} |g_{j_m} \alpha_{j_m}| \geq \frac{1}{2} |g_{j_m} d_{j_m}| \geq \frac{1}{2n} \chi.$$

Combining this bound with (3.6) and (3.7), we thus obtain that (2.78) holds with $\kappa_{\text{red}} = 1/2n^2$. □

One can now ask if the smoothing effect of the relaxation schemes used in the multigrid methods is also present in the SCM method. This effect is shown on Figure 3.1 where the evolution of the error (the distance between the iterate and the solution) is presented. We clearly see that the same effect appears on the error of the problem.

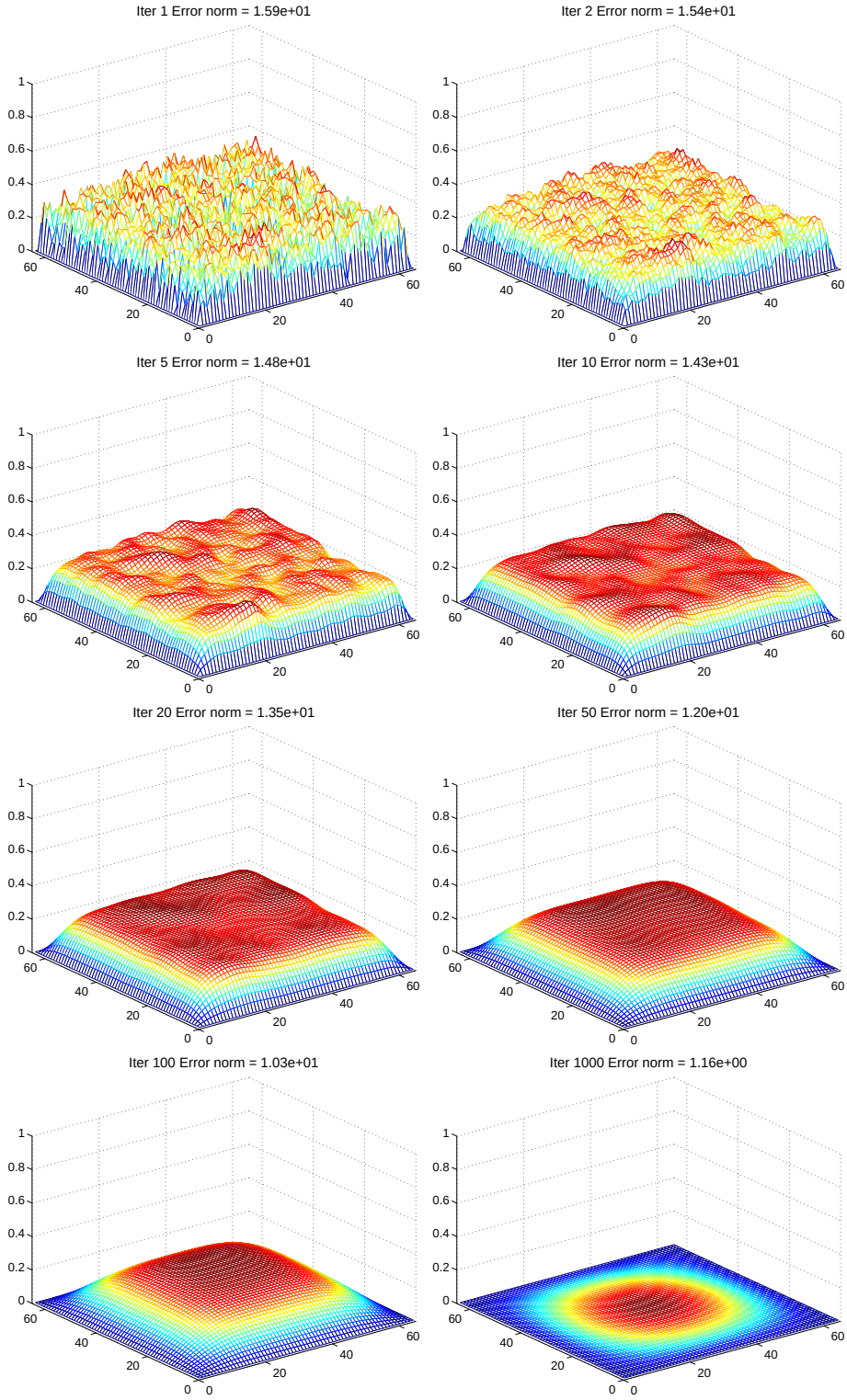

Figure 3.1: Illustration of the smoothing effect on the error of a quadratic problem.

3.1.2 Linesearch

The implementation whose numerical performance is discussed in the next Chapter uses a version that combines the traditional trust-region techniques with a linesearch, in the spirit of Toint (1983, 1987), Nocedal and Yuan (1998) and Gertz (1999) (see Conn et al., 2000, Section 10.3.2). More precisely, if $\rho_{i,k} < \eta_1$ in Step 4 of Algorithm RMTR$_\infty$ and the step is *gradient related* in the sense that

$$|\langle g_{i,k}, s_{i,k}\rangle| \geq \epsilon_{\text{gr}} \|g_{i,k}\|_2 \, \|s_{i,k}\|_2$$

for some $\epsilon_{\text{gr}} \in (0, 1)$, the step corresponding to a new iteration and a smaller trust-region radius can be computed by backtracking along $s_{i,k}$, instead of recomputing a new one using SCM smoothing. On the other hand, if some iteration at the topmost level is successful and the minimizer of the quadratic model in the direction $s_{r,k}$ lies sufficiently far beyond the trust-region boundary, then a single doubling of the step is attempted to obtain further descent, a strategy reminiscent of the *internal doubling* procedure of Dennis and Schnabel (1983) (see Conn et al., 2000, Section 10.5.2), or the *magical step* technique of Conn, Vicente and Visweswariah (1999) and Conn et al. (2000), Section 10.4.1. The theoretical arguments developed in these references guarantee that global convergence of the modified algorithm to first-order critical points is not altered.

3.1.3 Second-order and Galerkin models

The gradient correction v_{i-1} in (2.65) ensures that h_i and h_{i-1} coincide at first order (up to the constant σ_i) in the range of the prolongation operator, since

$$\langle g_{i,k}, P_i s_{i-1}\rangle = \frac{1}{\sigma_i}\langle R_i g_{i,k}, s_{i-1}\rangle = \frac{1}{\sigma_i}\langle g_{i-1,0}, s_{i-1}\rangle.$$

Although this feature is theoretically crucial, our experience indicates that is not enough to obtain an efficient numerical method. We can also achieve coherence of the second-order models by choosing

$$h_{i-1}(x_{i-1,0} + s_{i-1}) = f_{i-1}(x_{i-1,0} + s_{i-1}) + \langle v_{i-1}, s_{i-1}\rangle + \tfrac{1}{2}\langle s_{i-1}, W_{i-1} s_{i-1}\rangle, \tag{3.8}$$

where $W_{i-1} = R_i \nabla^2 h_i(x_{i,k}) P_i - \nabla^2 f_{i-1}(x_{i-1,0})$, since we then have that

$$\langle P_i s_{i-1}, \nabla^2 h_i(x_{i,k}) P_i s_{i-1,}\rangle = \frac{1}{\sigma_i}\langle s_{i-1}, \nabla^2 h_{i-1}(x_{i-1,0}) s_{i-1}\rangle.$$

The second-order model (3.8) is of course more costly, as the matrix W_{i-1} must be computed when starting the minimization at level $i-1$ and must also be used to update the gradient of h_{i-1} at each successful iteration at level $i-1$.

Another strategy consists of choosing $f_{i-1}(x_{i-1,0}+s_{i-1}) = 0$ for all s_{i-1} in (3.8). This strategy amounts to considering the lower-level model as the "restricted" version of the quadratic model at the upper level (this is known as the *Galerkin approximation*) and is interesting in that no evaluation of f_{i-1} is required. In the unconstrained case, when this model is strictly convex

and the trust region is large enough, one minimization in Algorithm RMTR$_\infty$ (without premature termination) corresponds to applying a Galerkin multigrid linear solver on the associated Newton's equation. Note that this choice is allowed within the theory presented in Gratton et al. (2008*a*), since the zero function is obviously twice-continuously differentiable, bounded below and has uniformly bounded Hessians.

3.1.4 Hessian of the models

Computing a model Hessian $H_{i,k}$ is often one of the heaviest tasks in Algorithm RMTR$_\infty$. Our choice in the experiments described in Chapter 4 is to use the exact second derivative matrix of the objective functions f_i. However, we have designed an automatic strategy that avoids recomputing the Hessian at each iteration when the gradient variations are still well predicted by the available $H_{i,k-1}$. More specifically, we choose to recompute the Hessian at the beginning of iteration (i,k) $(k > 0)$ whenever the preceding iteration is not successful enough (i.e. $\rho_{i,k-1} < \eta_H$) or when (since it indicates that the Hessian approximation is relatively poor)

$$\|g_{i,k} - g_{i,k-1} - H_{i,k-1}s_{i,k-1}\|_2 > \epsilon_H \|g_{i,k}\|_2,$$

where $\epsilon_H \in (0,1)$ is a small user-defined constant. Otherwise, we use $H_{i,k} = H_{i,k-1}$. Default values of $\epsilon_H = 0.15$ and $\eta_H = 0.5$ appear to give satisfactory results in most cases and these are the values we use in our reported tests.

3.1.5 Prolongations and restrictions

We have chosen to define the prolongation and restriction operators P_i and R_i as follows. The prolongation is chosen as the *linear interpolation* operator, and the restriction is its transpose normalized to ensure that $\|R_i\|_\infty = 1$ and $\sigma_i = \|P_i\|_\infty^{-1}$ (see (2.64)). These operators are never assembled, but are rather applied locally for improved efficiency. Cubic interpolation could also be used in principle, but it produces denser Galerkin models and is very restrictive in the context of Gelman-Mandel restrictions. Moreover our experience is that the algorithm is computationally less efficient.

3.1.6 Free and fixed form recursions

An interesting feature of the RMTR$_\infty$ framework is that its convergence properties are preserved if the minimization at lower levels ($i = 0, \ldots, r-1$) is stopped after the *first successful iteration*. The flexibility of this allows us to consider different recursion patterns, namely *fixed-form* and *free-form* ones. In a fixed form recursion pattern, a maximum number of successful iterations at each level is specified (like in V- and W-cycles in multigrid algorithms, see Briggs et al. (2000)). If no such premature termination is used but the minimization at each level is carried out until one of the classical termination conditions on the criticality measure and step size (see Step 5 of Algorithm RMTR$_\infty$) is satisfied, then the actual recursion pattern is uniquely determined by the progress of minimization at each level (hence yielding a free form recursion pattern).

In the next Chapter, we compare three recursion forms. In the first form, which we call the V-form, the minimization at the lower levels consists of one successful smoothing iteration, followed by a successful recursive iteration, itself followed by a second successful smoothing iteration[(1)]. The second form is called W-form and is defined as a V-form to which is added one successful recursive iteration, and a final smoothing iteration. The third form is the free form recursion as explained above, in which, however, we impose that smoothing iterations and recursive (successful) iterations alternate at all levels but the coarsest. Indeed, during our experiments, we have found this alternance very fruitful (and rather natural in the interpretation of the algorithm as an alternance of high frequency reductions and low frequency removals).

Note that for each recursion form, any remaining iteration is skipped if one of the termination conditions in Step 5 of Algorithm RMTR$_\infty$ is satisfied.

3.1.7 Computing the starting point at the fine level

We also take advantage of the multilevel recursion idea to compute the starting point $x_{r,0}$ at the finest level by first restricting the user-supplied starting point to the lowest level and then applying Algorithm RMTR$_\infty$ successively at levels 0 up to $r-1$. In our experiments based on regular meshes (see Chapter 4), the accuracy of the criticality measure that is required for termination at level $i < r$ is given by

$$\epsilon_i^\chi = \epsilon_{i+1}^\chi \sigma_{i+1}, \tag{3.9}$$

where ϵ_r^χ is the user-supplied criticality requirement for the topmost level and σ_{i+1} is due to the definition (2.73) of the criticality measure and the fact that (2.66) yields this constant as the ratio between two linearized decreases at successive levels. Once computed, the solution at level i is then prolongated to level $i+1$ using *cubic interpolation*. The criteria (3.9) comes from the fact that we want that the prolongation of our step stay critical for the upper level $i+1$ excepted for the highest frequencies of the error that are not visible at level i and only appear at level $i+1$ and finer levels.

3.1.8 Constants choice and recursive termination thresholds

We conclude the description of our practical algorithm by specifying our choice for the constants and the level-dependent criticality thresholds ϵ_i^χ. We set

$$\kappa_\chi = 1/4, \quad \eta_1 = 0.01, \quad \eta_2 = 0.95, \quad \gamma_1 = 0.05 \text{ and } \gamma_2 = 1.00, \tag{3.10}$$

as this choice appears most often appropriate. The value 1 is also often satisfactory for the $\Delta_{i,0}$. We considered two possible expressions for the criticality thresholds. The first is related to the descent condition (2.74) and is given by

$$\epsilon_i^\chi = \kappa_\chi \chi_{i,k} \sigma_{i+1}. \tag{3.11}$$

[(1)] A the coarsest level, 0, smoothing iterations are skipped and recursion is impossible.

We also considered using (3.9), but this was found to be unsuitable for recursive iterations. Indeed, it often prevented the effective use of coarse level computations because it was satisfied at $x_{0,i}$, resulting in an immediate return to the fine level. We thus considered an adaptation of this rule given by

$$\epsilon_i^\chi = \min\{\epsilon_{i+1}^\chi, \kappa_\chi \chi_{i,k}\}\sigma_{i+1}. \tag{3.12}$$

This adaptation was further motivated by the observation that the alternance between SCM smoothing and recursive iterations is very efficient in practice and we want thus to impose that at least one lower-level iteration is done if the descent condition (2.74) allows it.

3.2 The RMTR package for multilevel optimization

The results obtained by the method are, as we will see in the next Chapter, excellent and suggest that the method can be of interest more widely. It was thus decided to construct a software implementing the RMTR$_\infty$ algorithm. The package, called RMTR, has been programmed in Fortran 95 and has been made available to the community as a part of the GALAHAD library of packages for solving nonlinear optimization problems (see Gould, Orban and Toint (2003*b*)). The areas covered by this library are unconstrained and bound-constrained optimization, quadratic programming, nonlinear programming, systems of nonlinear equations and inequalities, and nonlinear least squares problems.

The library is available at

`http://galahad.rl.ac.uk/index.html`

and the conditions of its use may be found at

`http://galahad.rl.ac.uk/cou.html`.

RMTR features a flexible strategy to accommodate different classes of discretized optimization problems. On one hand the user is allowed to provide the algorithm his/her own general multilevel information, and, on the other hand, RMTR also provides special facilities for problems defined on regular (geometric) discretization grids, where each f_{i-1} is a restriction of f_i on a coarser grid. The details of how these options may be used are covered below.

The information about the hierarchy may be difficult to obtain. Thus, RMTR features algorithms that only need the information about the finest level (the All on Finest algorithm, the Multilevel on Finest algorithm and the Full Multilevel on Finest algorithm). Although, since the knowledge of the hierarchy may lead to better numerical results, RMTR also features algorithms that take advantage of this hierarchy (the Mesh Refinement algorithm and the Full Multilevel algorithm). For more informations about these algorithms and a numerical comparison of these ones, see (Gratton, Mouffe, Sartenaer, Toint and Tomanos 2009). The complete documentation of the package may be found in Appendix D and a summary of this documentation is now described.

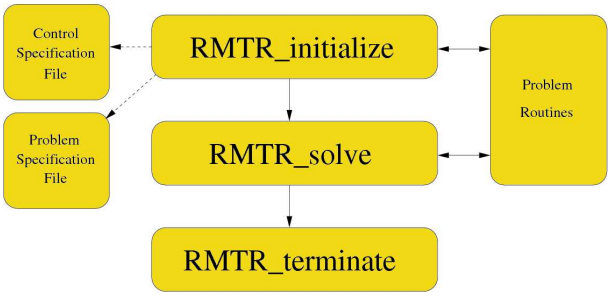

Figure 3.2: Calling sequence of the RMTR package

The implementation of the package is based on three structures that hold all the information about the progress of the algorithm:

- The first structure is called `RMTR_problem_type` and is used to hold all the information about the multilevel structure of the problem. It is a double linked chained list where each component of the structure contains all the information about a particular level i (the level index, the number of variables at this level, the current iterate and the derivatives at this iterate, the bounds and the transfer operators to the coarser level and to the finer level) and two pointers to the coarser level and to the finer level.

- The second structure `RMTR_control_type` is used to hold controlling data. All the parameters defining the method are hold in this structure (the descent parameter, the trust-region parameters, the maximum number of smoothing cycles, ...)

- The third structure `RMTR_inform_type` is used to hold parameters that give information about the progress and needs of the algorithm.

Access to the package requires a `USE` statement such as

```
USE RMTR
```

The package uses two specification files. The first one is a control file whose purpose is to modify the default values of the algorithmic parameters of the method. The second file defines important characteristics of the problem, such as its dimension, ... (the complete list of the problem parameters is described in Appendix D, Section D.1.5.2). The description and syntax of these specification files is available in Appendix D, Section D.1.7.

The actual call to the package consists of three successive subroutine calls, as illustrated in Figure 3.2.

1. First, the subroutine `RMTR_initialize` is used to allocate memory to the structures, to give default values to the parameters of the method and to read the specification files to modify the values of these default parameters. It is called by the statement

```
CALL RMTR_initialize( control,inform,problem,algospecs,
  probspecs [, MY_GRAD=GRAD] [, MY_STRUCT=STRUCT]
  [,MY_SPARSITY=SPARSITY] [, MY_LOWER_BOUND=LOWER_BOUND]
  [,MY_UPPER_BOUND=UPPER_BOUND]
  [,MY_PROLONGATION=PROLONGATION]
  [,MY_RESTRICTION=RESTRICTION] [, MY_SIZE=SIZE] )
```

where `control` is a scalar argument of type `RMTR_control_type`,`inform` is a scalar argument of type `RMTR_inform_type`, `problem` is a pointer of type `RMTR_problem_type` and `algospecs` and `probspecs` are two strings containing the name of the control and problem specification files respectively. The other arguments are optional and are used to provide the algorithm routines describing the problem (`GRAD` is the user subroutine to compute the objective gradient, `STRUCT` is the user subroutine to compute the Hessian structure, ...). They are all described more in details in Section D.1.5. On exit, `control` contains default values for all the components that may be changed by the values given by the user in the control specification file and all components of the multilevel structure are allocated according to the values provided by the user in the problem specification file.

2. Then the subroutine `RMTR_solve` is used to solve the optimization problem. We first compute or read the starting point of the algorithm (see Section D.1.4) and then it apply the algorithm. the routine is called by

```
CALL RMTR_solve( control, inform, problem , MY_FUN=FUN ,
  MY_GRAD=GRAD [, MY_HESS=HESS] )
```

where `control` is a scalar argument of type `RMTR_control_type`,`inform` is a scalar argument of type `RMTR_inform_type`, `problem` is a pointer of type `RMTR_problem_type`. The other arguments are routines to describe the problem. `FUN` is the user subroutine to compute the objective function `GRAD` is the user subroutine to compute the objective gradient and `HESS` is the user subroutine to compute the objective Hessian and is optional. The headers of these routines are described in Section D.1.5. This routine is called to solve the problem by applying the RMTR algorithm. A call to this routine must be preceded by a call to the `RMTR_initialize` routine. On exit, the solution is in the `problem%x` component. A successful call to the routine `RMTR_solve` is indicated when the component `status` of the `inform` argument has the value 0. For other return values of the `status` component, see Section D.1.6. The other `inform` components contain the iterations history of the algorithm as explained in Section D.1.3.3.

3. Finally, the subroutine `RMTR_terminate` is used to automatically deallocates the RMTR-specific arrays, to write the solution in a file if required by the user and to print a summary of the iterations if required by the user. It is called by

```
CALL RMTR_terminate( control, inform, problem )
```

where `control` is a scalar argument of type `RMTR_control_type`,`inform` is a scalar argument of type `RMTR_inform_type`, `problem` is a pointer of type `RMTR_problem_type`. A call to this routine must be preceded by a call to the `RMTR_initialize` routine.

Then the application of the algorithm on a problem simply consists in a successive call of these three routines. An example is presented in Section D.2. The complete description of the package containing the specification of the three RMTR-specific structures, a full description of the RMTR routines and of the routines describing the problem are also presented in Appendix D.

Chapter 4

Numerical experiments

4.1 Test problems

We have considered a suite of minimization problems in infinite-dimensional spaces, involving differential operators. These problems are detailed in Appendix B. The differential operators are discretized on a hierarchy of regular grids such that the coarse grid at level $i-1$ is defined by taking every-other point in the grid at level i: the ratio between the grid spacing of two consecutive levels in each coordinate direction is therefore 2. The grid transfer operators P_i are defined as in classical geometric multigrid settings, using interpolation operators. The restriction operators R_i are such that (2.64) holds.

All experiments discussed below consider the solution of the test problem on the finest grid, whose size may be found in Table 4.1, together with other problems characteristics. The algorithms were terminated when the criticality measure (2.73) at the finest level was below 10^{-3} for all the test cases. Notice that requiring that $\chi_{r,k} \leq \epsilon_r = 10^{-3}$ is approximately the same as requiring the *scaled criticality measure* $\frac{\chi_{r,k}}{n_r}$, whose value is comparable, for example, with the infinity norm of the projected gradient $||Pr_{\mathcal{F}}(x_{r,k} - g_{r,k}) - x_{r,k}||_\infty$, to be such that $\frac{\chi_{r,k}}{n_r} \leq \frac{\epsilon_r}{n_r}$. This last tolerance is, for instance, $\frac{\epsilon_r}{n_r} \approx 10^{-9}$ in the case where $n_r = 1046529$ and $\frac{\epsilon_r}{n_r} \approx 10^{-8}$ if $n_r = 65025$.

Our testing strategy, which is discussed in the next paragraphs, is first to establish a good default value for the algorithmic parameters, and, in a second step, to compare the resulting method with other competing approaches.

4.2 In search of efficient default parameters

Given the relatively large number of parameters in our method, a complete discussion of all possible combinations is outside the scope of this thesis. We have therefore adopted the following approach. We first fixed the parameters for which a reasonable consensus already exists, namely the trust-region parameters η_1, η_2, γ_1 and γ_2, which are set as in (3.10), in accordance with Conn et al. (2000) and Gould, Orban, Sartenaer and Toint (2005). The initial trust-region radii $\Delta_{i,0}$ are set to 1, as suggested in Section 17.2 of the first of these references. A second class of parameters was then isolated, containing algorithmic options with very marginal effect

Problem name	n_r	r	Comment
DNT	511	8	1-D, quadratic
P2D	1046529	9	2-D, quadratic
P3D	250047	5	3-D, quadratic
DEPT	1046529	9	2-D, quadratic, (Minpack 2)
DPJB	1046529	9	2-D, quadratic, with bound constraints, (Minpack 2)
DODC	65025	7	2-D, convex, (Minpack 2)
MINS-SB	1046529	9	2-D, convex, smooth boundary conds.
MINS-OB	65025	7	2-D, convex, oscillatory boundary conds.
MINS-DMSA	65025	7	2-D, convex, (Minpack 2)
IGNISC	65025	7	2-D, convex
DSSC	1046529	9	2-D, convex, (Minpack 2)
BRATU	1046529	9	2-D, convex, (Minpack 2)
MINS-BC	65025	7	2-D, convex, with bound constraints
MEMBR	393984	9	2-D, convex, free boundary, with bound constraints
NCCS	130050	7	2-D, nonconvex, smooth boundary conds.
NCCO	130050	7	2-D, nonconvex, oscillatory boundary conds.
MOREBV	1046529	9	2-D, nonconvex

Table 4.1: Test problem characteristics

on the computational results. These are the choice of activating the linesearch mechanism (we allow for backtracking if the initial step is unsuccessful and at most one extrapolation evaluation if it is successful and gradient-related with $\epsilon_{\text{gr}} = 0.01$), the parameters ϵ_H and η_H of the Hessian evaluation strategy (we chose $\eta_H = 0.5$ and $\epsilon_H = 0.15$), and the degree of the interpolation in the prolongation operator (linear interpolation is used within recursive iterations, and cubic interpolation when prolongating the solution at a coarse level into a starting point at the next finer one). The remaining algorithmic parameters were either central in the definition of our method or found to alter the performance of the method significantly, and we focus the rest of our discussion on their choice.

We begin by determining the optimal combination of these parameters. For this purpose, we ran a large number (192) of possible combinations of these options on our set of 17 test problems and report all results of the 3264 runs on a comet-shape graph representing a measure of the effort spent in function evaluations as a function of CPU-time. More precisely, we have first scaled, separately for each test problem, the number of function evaluations and CPU-time by dividing them by the best obtained for this problem by all algorithmic variants. We then plotted the averages of these scaled measures on all test problems for each algorithmic variant separately, after removing the variants for which the CPU limit of 1000 seconds was reached on at least one problem. In the first of these plots (Figures 4.1 and 4.2), we have used triangles for variants where the coarse Galerkin model is chosen at recursive iterations and stars for variants where the second-order model (3.8) is chosen instead[(1)].

[(1)]Notice that we did not represent the tests where the coarse model is defined as in (2.65) because preliminary

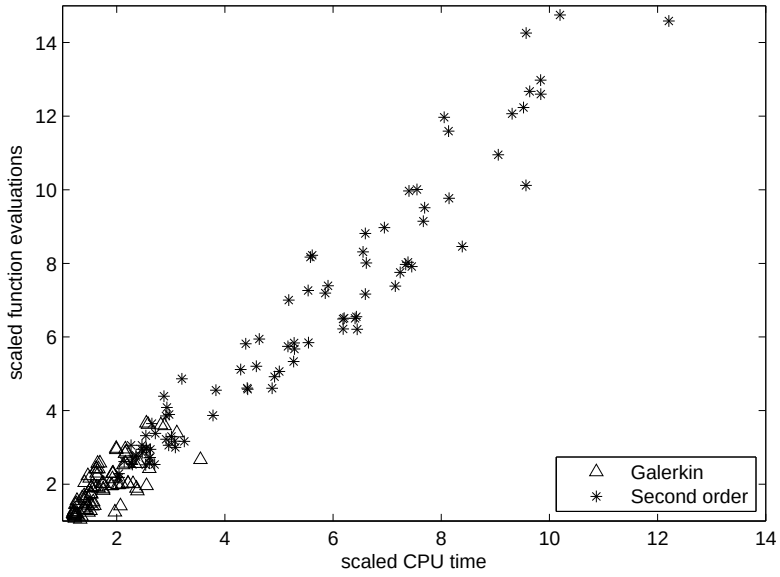

Figure 4.1: Average scaled function evaluations versus average scaled CPU-time for all algorithmic variants, distinguishing the type of model used.

We note a substantial spread of the results, with some options being up to fifteen times worse than others. The worst cases (in the top right corner) correspond to combinations of the quadratic model (3.8) with a single smoothing cycle and small values of κ_χ. On the other hand, the choice of the Galerkin model is very clearly the best. This is mainly due to the numerical cost of the alternative because it requires a function/Hessian evaluation and a matrix update for each model in (3.8). Even on the test cases for which this choice proves superior in number of iterations, the advantage is then lost in CPU-time. In view of this conclusion, we therefore select the the Galerkin model as our default and restrict further analysis to this case.

We now consider the number of smoothing cycles performed at each Taylor iteration (at a level $i > 0$) and illustrate our results in Figure 4.3. All algorithmic variants (with the coarse Galerkin model) are again represented in a picture similar to Figure 4.1, where different symbols are used to isolate variants using different number of smoothing cycles.

An important property of this option is that the number of function evaluations decreases as the number of cycles increases, because a single evaluation is exploited to a fuller extent if more cycles are performed consecutively. This correlation is maintained up to a level (probably depending on the quadraticity of the objective function) beyond which the work of additional cycles is no longer effective. The correlation is much less clear when considering CPU-time, even if our result indicate that too few smoothing cycles is seldom the best option. Good choices seem to range between 2 and 7 cycles.

tests showed that performing only a first-order correction is indisputably not competitive.

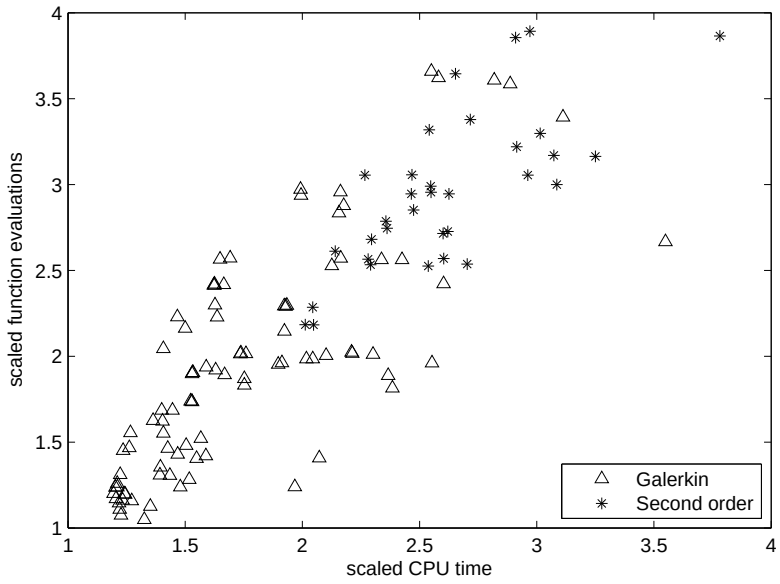

Figure 4.2: Detail of the lower left-hand corner of Figure 4.1.

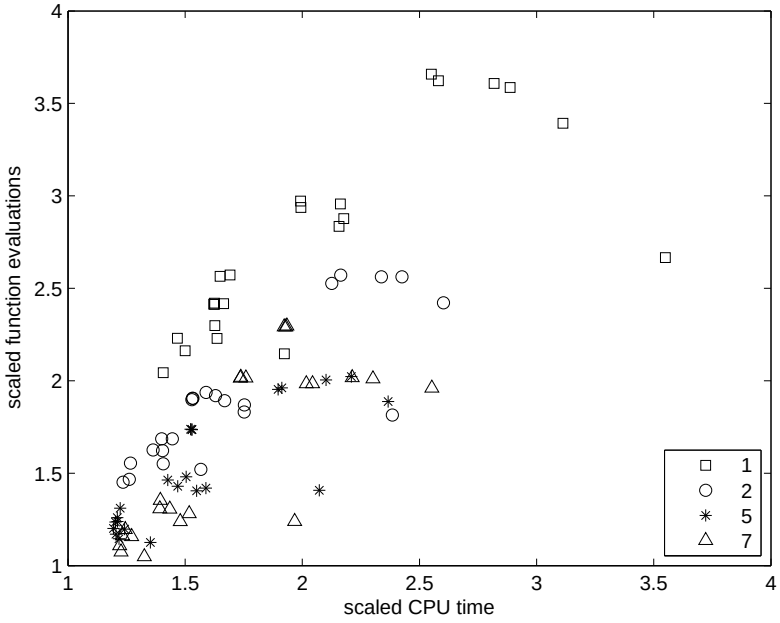

Figure 4.3: Average scaled function evaluations versus average scaled CPU-time for all algorithmic variants, distinguishing the number of smoothing cycles per Taylor iteration.

Choosing between the values for κ_χ is not easy. We have considered four possible values (1/2, 1/4, 1/8, 1/16). We first note that choosing κ_χ to be significantly larger that 1/2 results in a poor exploitation of the multilevel nature of the problem, since recursive iterations become much less frequent. On the other hand, values much smaller than 1/16 are also problematic because recursive iterations are then initiated for a too marginal benefit in optimality, although

this strategy is closer to the unconditional recursive nature of multigrid algorithms for linear systems. In our tests the best threshold has been obtained for either $\kappa_\chi = 1/2$ or $\kappa_\chi = 1/4$, with a slight advantage for the second choice (see Figure 4.4, which is built on the same principle as the previous ones).

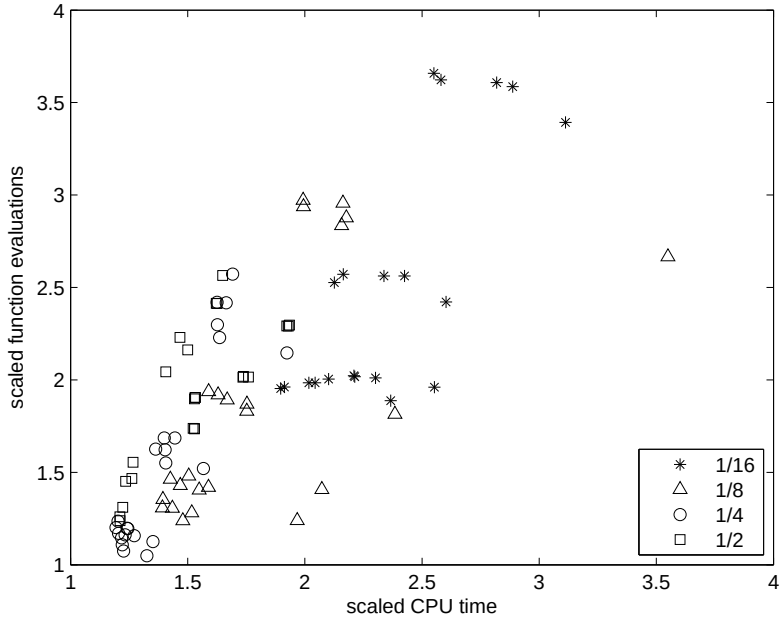

Figure 4.4: Average scaled function evaluations versus average scaled CPU-time for all algorithmic variants, distinguishing the values of κ_χ.

We now turn to the impact of the cycle types on performance, which is illustrated in Figure 4.5. Remarkably, an excellent performance can be obtained with the three considered cycle styles, quite independently of the other algorithmic parameters. In particular, this indicates that the strategy for automatically adapting the cycle type to the problem at run-time is reasonably efficient. It is however slightly more complicated and the simpler V-form may often be preferred in practice.

Finally, Figure 4.6 shows the effect of the coarse criticality threshold choice between (3.11) (nomin) and (3.12) (min). It indicates that (3.12) is generally preferable, although the performance remains mixed.

As a conclusion of this analysis, we decided to select the defaults as the use of the Galerkin model, 7 smoothing cycles per Taylor iteration, a value of $\kappa_\chi = 1/4$, V-form iterations and the (3.12) termination rule.

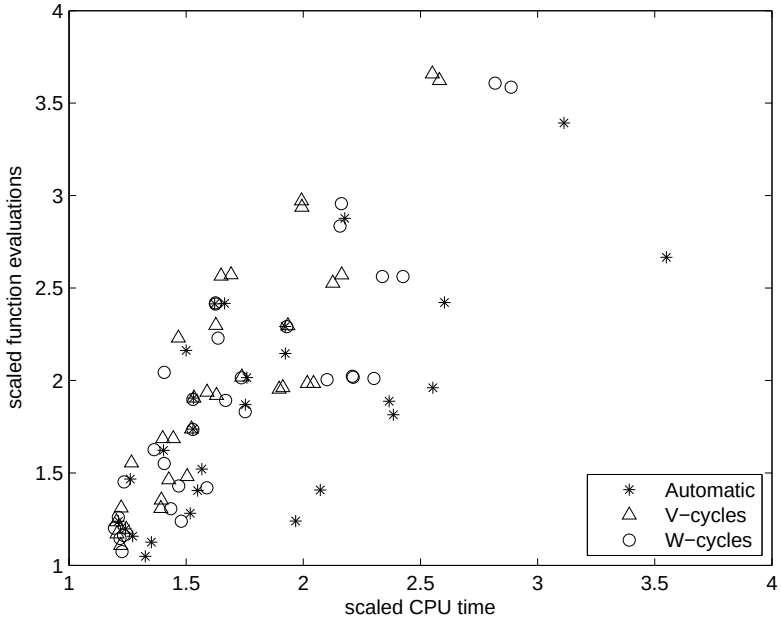

Figure 4.5: Average scaled function evaluations versus average scaled CPU-time for all algorithmic variants, distinguishing the type of recursive cycles.

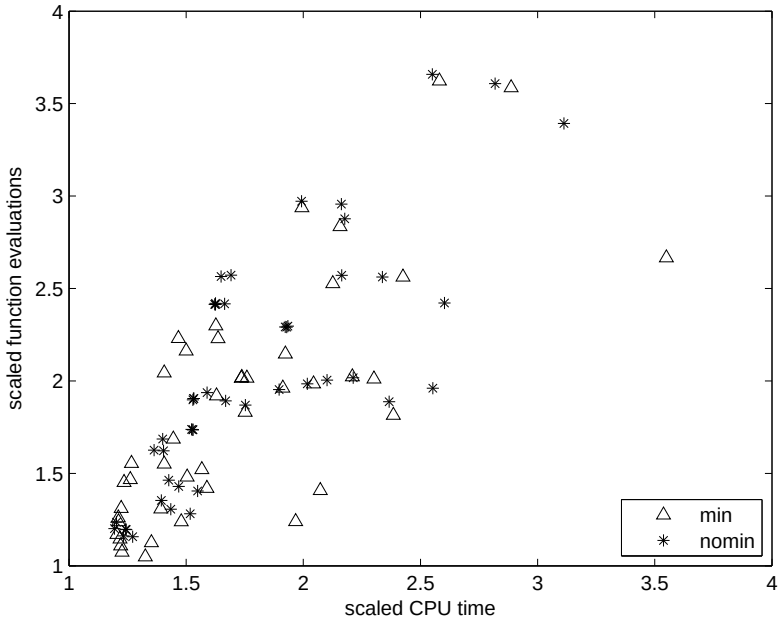

Figure 4.6: Average scaled function evaluations versus average scaled CPU-time for all algorithmic variants, distinguishing the type of lower level criticality threshold.

4.3 Performance of RMTR$_\infty$

We now analyze the performance of the resulting recursive trust-region algorithm in comparison with other approaches on our battery of 17 test problems. This analysis is conducted by comparing four algorithms:

- the *all on finest* (**AF**) algorithm, which is a standard Newton trust-region algorithm (with PTCG as subproblem solver) applied at the finest level, without recourse to coarse-level computations;

- the *mesh refinement technique* (**MR**), where the discretized problems are solved from the coarsest level (level 0) to the finest one (level r) successively, using the same standard Newton trust-region method, and where the starting point at level $i+1$ is obtained by prolongating (using P_{i+1}) the solution obtained at level i;

- the *multilevel on finest* (**MF**) method, where Algorithm RMTR$_\infty$ is applied directly on the finest level;

- the *full multilevel* (**FM**) algorithm where Algorithm RMTR$_\infty$ is applied successively on progressively finer discretizations (from coarsest to finest) and where the starting point at level $i+1$ is obtained by prolongating (using P_{i+1}) the solution obtained at level i.

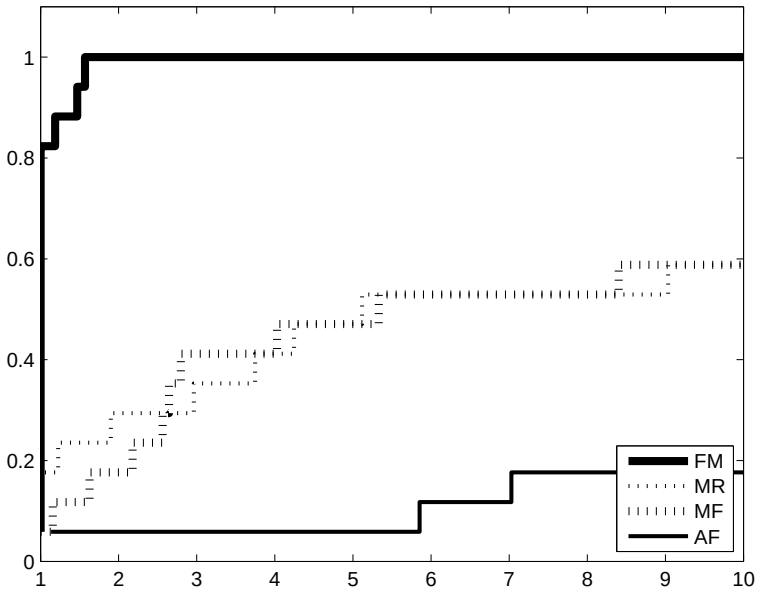

Figure 4.7: Performance profile for CPU time with variants AF, MF, MR and FM (17 test problems).

A CPU-time performance profile is presented in Figure 4.7 for all our test problems and these four variants. The first conclusion is that the full multilevel variant (FM) clearly outperforms all other variants. The second observation is that the AF variant is, as expected, by far the worst. The remaining two variants are surprisingly close, and the use of recursive iterations on the fine level appears to have an efficiency similar to that of optimizing on successively finer grids. These observations are confirmed by a detailed analysis of the complete numerical results presented in Appendix C.

4.3.1 Unconstrained problems

The conclusions of the previous paragraph do not tell the whole story, as we may be interested to see if the gain in performance obtained is indeed the result of a multigrid-like gain in efficiency. To answer this question, we now turn to a more detailed comparison of the MR and FM variants on three specific unconstrained test problems (P2D, MINS-SB and NCCS), which we consider representative of the various problem classes mentioned in Table 4.1.

The performance of the algorithms is illustrated for each of these problems by a figure showing the history of the scaled criticality measure defined in Section 4.1 when the MR (thin line) and the FM (bold line) algorithms are used. In these figures, the dashed line represents the increase of the scaled criticality measure when a solution is prolongated during the application of a mesh refinement process. Moreover, and because iterations at coarse levels are considerably cheaper than those at higher ones, we have chosen to represent these histories as a function of the *equivalent number of finest iterations*, given by

$$q = \sum_{i=0}^{r} q_i \left(\frac{n_i}{n_r} \right), \tag{4.1}$$

where q_i is the number of iterations at level i.

We first consider the quadratic minimization problem P2D in Figure 4.8. Because this problem is equivalent to solving a linear system of equations, we expect algorithm FM to exhibit a multigrid-type behavior. Looking at Figure 4.8, we see that this is effectively the case. We note that FM is considerably more efficient than MR (by a factor approaching 100). This last result confirms that our trust-region globalization is not hindering the known efficiency of the multigrid methods for this type of problems. Note that the significant increase of the scaled criticality measure when a lower level solution is prolongated to an upper level starting point is due to the fact that oscillatory components of the error cannot be represented on the coarser levels and therefore could not have been reduced at these levels.

The same conclusions seem to apply when we consider Figures 4.9[(2)] and 4.10, where the same algorithms are tested on MINS-SB and NCCS, respectively. This is remarkable because the problems are now more general and do not correspond anymore to linear systems of equations (MINS-SB is nonquadratic) or elliptic problems (NCCS is non-convex).

An important feature of the classical trust-region algorithm is that its convergence is speeded up when the trust-region becomes inactive (because the algorithm then reduces to Newton's

[(2)] Observe that the MR variant had to be stopped after 1 hour of computing on this problem.

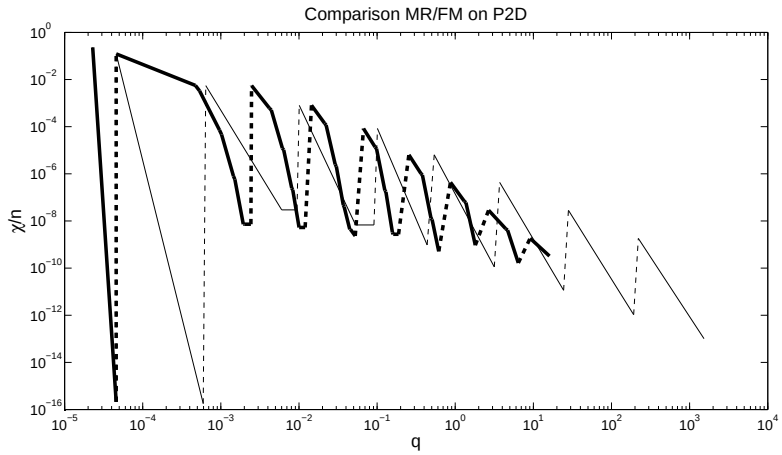

Figure 4.8: History of the scaled criticality measure on P2D. A small circle surrounds the iterations where the trust region is active. *Note that both axis are logarithmic.*

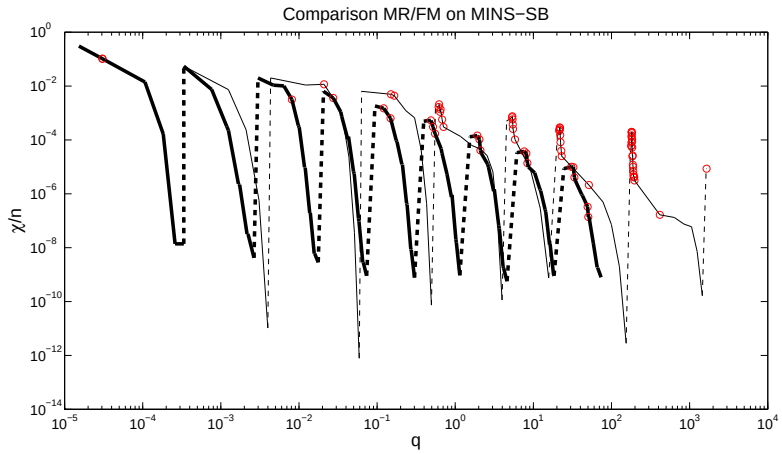

Figure 4.9: History of the scaled criticality measure on MINS-SB. A small circle surrounds the iterations where the trust region is active. *As above, both axis are logarithmic.*

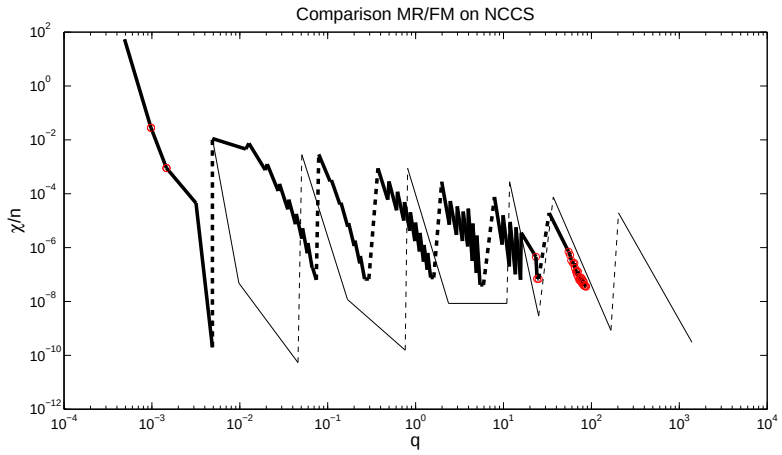

Figure 4.10: History of the scaled criticality measure on NCCS. A small circle surrounds the iterations trust region is active.*As above, both axis are logarithmic.*

method and thus achieves quadratic convergence under the assumption that the second-order Taylor model (2.62) is chosen). Iterations where the trust-region is active have been indicated, in the above figures, by a small circle (observe that they often correspond to non-monotonic decrease of the scaled criticality). We note that no such iteration occurs for MR and FM on P2D, and also that convergence speeds up for all methods as soon as the trust region becomes inactive, even if the rate is at most linear for the multilevel methods.

4.3.2 Bound-constrained problems

We finally evaluate the RMTR$_\infty$ algorithm on the bound-constrained problems DPJB, MINS-BC and MEMBR. The results for these problems are presented in Figures 4.11 to 4.12.

We first note that the relative performance of the considered algorithms is very similar to that already analyzed for unconstrained problems, at least for DPJB[(3)] and MEMBR. On this last problem, the figure indicates that further efficiency gains could be obtained by a finer tuning of the termination accuracy at levels 5, 6 and 7. On all three problems, a gain in CPU time of a factor exceeding 10 is typically obtained when considering the multilevel variant. Again, the trust-region constraint is mostly inactive on these examples. This is in sharp contrast with MINS-BC, where it plays an important role, except in the asymptotics (as expected from trust-region convergence theory).

[(3)]We should note here that the Hessian of this quadratic problem is not supplied by the MINPACK code and has been obtained once and for all at the beginning of the calculation by applying an optimized finite-difference scheme (see Powell and Toint, 1979).

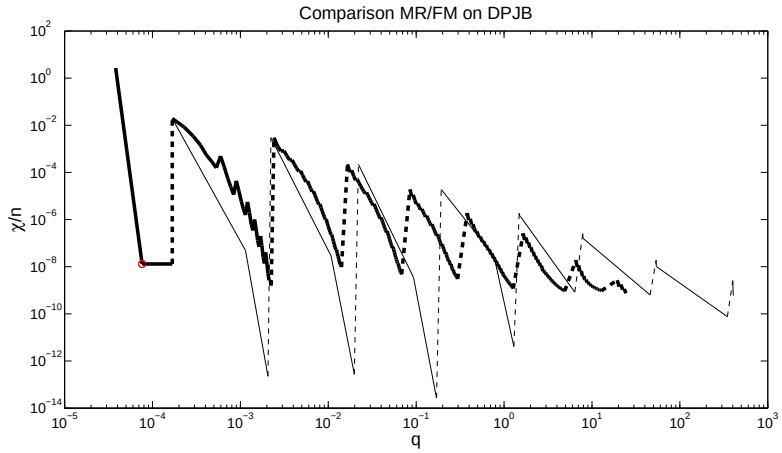

Figure 4.11: History of the scaled criticality measure on DPJB. *As above, both axis are logarithmic.*

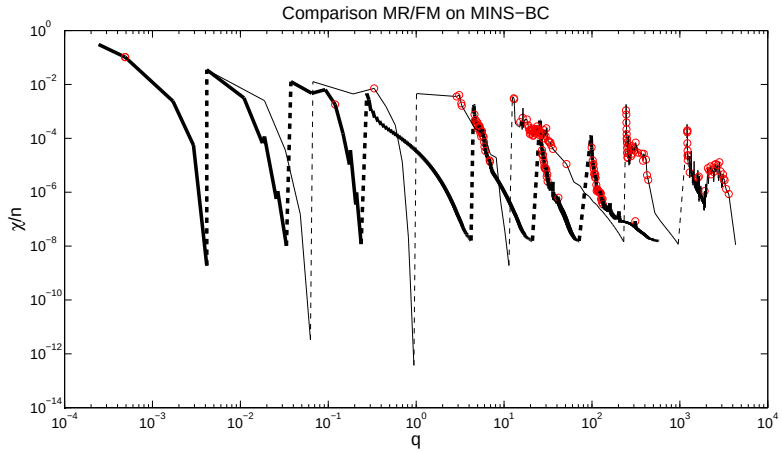

Figure 4.12: History of the scaled criticality measure on MINS-BC. A small circle surrounds the iterations where the trust region is active. *As above, both axis are logarithmic.*

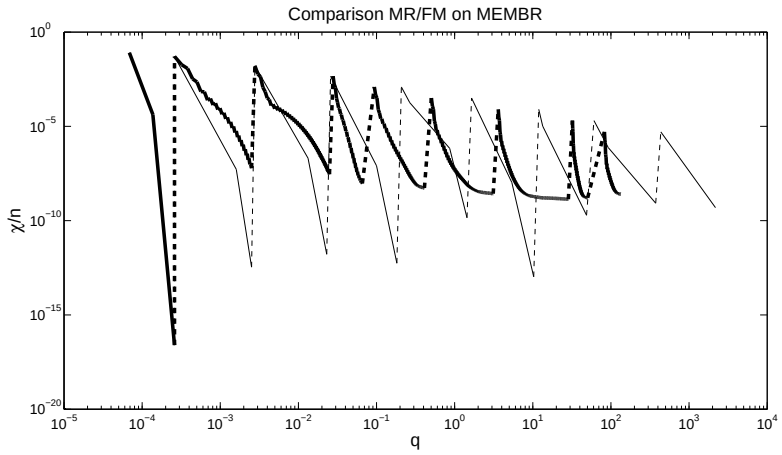

Figure 4.13: History of the scaled criticality measure on MEMBR. *As above, both axis are logarithmic.*

4.4 The design of progressive adaptive lenses

INDO is a Spanish industry specialized in the design of progressive adaptive lenses. They have submitted us this application that states the design of the lens as an optimization problem.

As people become older, their vision decreases; that is, it becomes more difficult for them to see objects at a short distance. Moreover, this problem may be combined with other optical problems like myopia that prevent a clear vision of far objects. To correct these problems, the user has three possibilities. First, he could use two different glasses to correct the two different problems. Although, this solution is very unpractical. Second, the user can use bifocal glasses, where an half-moon part of the lens is dedicated to the vision of near objects while the rest of the lens is designed for the vision of far objects (see Figure 4.14). These glasses are also unpractical since the transition between the two zones is really abrupt. Third, the user can use progressive adaptive lenses (PAL). These lenses are characterized by an upper zone dedicated to the vision of far objects, a lower zone dedicated to near objects and the transition between the two zones is progressive to bother at least as possible the user (see Figure 4.14).

The third solution is clearly the most convenient for the user and is the one we are interested in. Mathematically, it is not simple to represent a PAL. Indeed, at the opposite of other lenses, it is impossible to use a piece of sphere to construct the lens. Moreover, to correct the user problem, we have to construct the lens with a given *optical power* at each point of the lens and the progressive modification of the lens corresponds to a progressive adaptation of the power along the lens. This adaptation produces inevitably an optical aberration called *astigmatism* that creates some blurring effect on the vision. The goal of the design of a progressive adaptive lens is thus to correct the user optical problems and to reject this blurring effect to the zones of the lens that are less used.

Let us formulate the problem mathematically. We denote by $S(u, v)$ the surface, by S_u, S_v its first derivatives with respect to the first and second variable respectively, and by S_{uu}, S_{uv} and

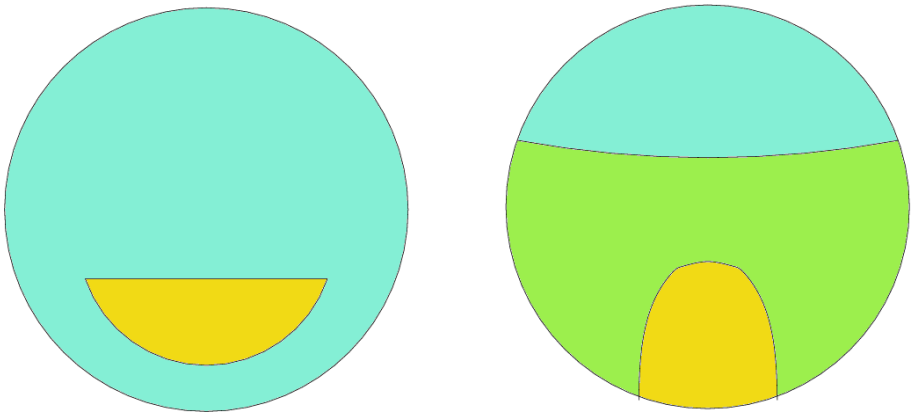

Figure 4.14: The left picture represents a bifocal lens and the right picture represents a PAL lens. We have represented in yellow the zone dedicated to the near vision, in green the zone dedicated to the intermediate vision and in blue the zone dedicated to the far vision.

S_{vv} its second derivatives. The optical power and astigmatism at a given point are given by

$$\begin{array}{rcl} Pow & = & \frac{\eta}{2}(k_1 + k_2), \\ Ast & = & \eta(k_1 - k_2), \end{array} \tag{4.2}$$

where η is a constant which depends of the material refractive index, and k_1 and k_2 are the maximum and minimum surface curvature at the point (also called principal curvatures). The interested reader may see O'Neill (1997) or Do Carmo (1976) for a good introduction to differential geometry and a complete explanation of these notions of curvature. Using the definitions of the maximum and minimum curvature, it is possible to derive a more practical definition of the astigmatism and optical power at a given point x_i,

$$Pow_i = 0.5 N_i^3 \eta \left((1 + S_u) S_{vv} + (1 + S_v) S_{uu} - 2 S_v S_u S_{uv} \right) \tag{4.3}$$

and

$$Ast_i^2 = 4 \left(\text{Pow}_i^2 - \eta^2 N_i^4 (S_{uu} S_{vv} - S_{uv}^2) \right), \tag{4.4}$$

where

$$N_i = (1 + S_u^2 + S_v^2)^{-\frac{1}{2}} \tag{4.5}$$

is the normal vector to the surface at the point x_i, $\eta = 523$ is a conversion in millimeters of the refractive index and where the derivatives are those computed at the point x_i.

The progressive adaptive lens design optimization problem is thus given by

$$\begin{array}{ll} \min & \sum_{i=1}^{M} Ast_i^2 \\ \text{s.t.} & PowMin_i \leq Pow_i \leq PowMax_i, \end{array} \tag{4.6}$$

where M is the number of points where we want to optimize the design of the PAL and $PowMin_i$ and $PowMax_i$ are lower and upper bound allowed on the optical power. This problem is really complicated. Indeed, both the objective function and the constraints are nonlinear

functions of the surface derivatives. A complete analysis of this problem and its solution by common optimization methods is presented in Fontdecaba i Baig (2000).

Since our multilevel algorithm only solves bound-constrained problems, we have reformulated the problem (4.6) as

$$\min \sum_{i=1}^{M} \alpha_i Ast_i^2 + \beta_i (Pow_i - PowRef_i)^2, \tag{4.7}$$

where α_i and β_i are given weights and $PowRef_i$ is the reference value for the optical power (that is the center of the interval $[PowMin_i, PowMax_i]$) represented in Figure 4.15.

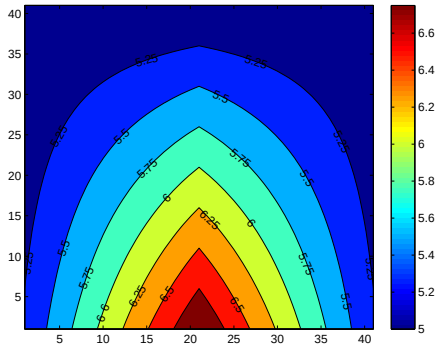

Figure 4.15: The reference power for the design of a progressive adaptive lens.

The only question that remains open to define the problem is the definition of the boundary of the problem. Actually, the boundary is not of practical importance. Indeed, when constructing the lens, engineers are only interested in the center part of the lens since the rest of the grid is not used to construct the lens. We have thus chosen to set it to zero.

We have launched the RMTR on a grid of 1 046 529 points. The results are presented in Figure 4.16 where we have represented the astigmatism and the error made on the optical power. The algorithm reaches the optimal solution within approximately 20 minutes. We observe that the values of the astigmatism becomes larger as we approach the boundary. We think this is due to the boundary condition we chose and we think that if we have a better boundary conditions better results could be obtained even in the middle of the PAL. Indeed, it seems that this boundary condition is not appropriate for the problem and add some stress on the lens that deteriorates the results.

We have thus chosen to try another boundary condition. Since we had no idea on what was the best boundary, we let the boundary completly free. This strategy provides us better results than other more clever ones like fixing one point of the surface. We have chosen a grid of 16 641 points for the finest level. We don't have try larger problems since the free boundary makes the problem very ill-conditioned and the software did not reach the optimal solution with the maximum of 10 000 iterations. Nevertheless, if we look at the solution obtained represented in Figure 4.18, we clearly see that the quality of the solution is better than in the first case and that the remaining error is close to the boundary. From our point of view, this is due to the

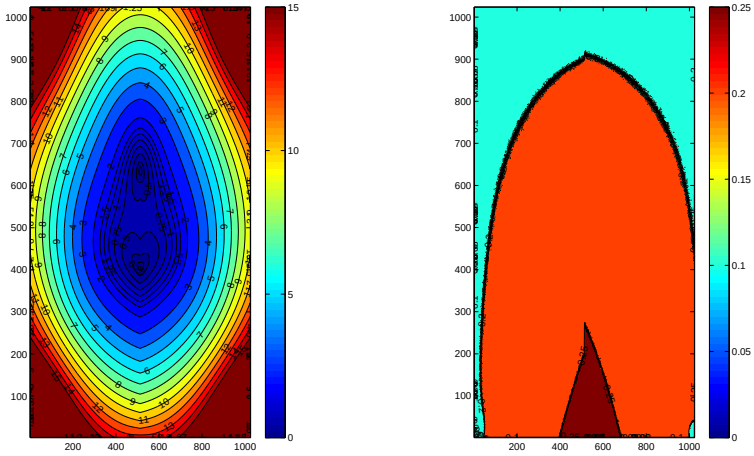

Figure 4.16: Results of the progressive adaptive lens design with a zero boundary. The left picture represents the astigmatism and the right picture represents the error made on the reference power.

fact than the exact boundary minimizing the stress on the surface is difficult to obtain by our minimization algorithm.

To overcome this problem we thus have chosen to modelize the problem otherwise. We approximate the surface as a polynomial of order p. We can note the surface as

$$S(u,v) \stackrel{\text{def}}{=} \sum_{i,j}^{i+j\leq p} c_{i,j} u^i v^j. \tag{4.8}$$

We minimize the same objective function (4.7) but now the variables are the polynomial coefficients and the exact surface derivatives may be obtained very easily. This leads to a simpler problem than the first one and moreover, the boundary condition is no more needed.

The multilevel structure of the problem is obtained by the degree of the polynomials and the restriction operator is simply the projection operator. In this case, a full multilevel algorithm is clearly not to use since the Hessian of the problem is full and thus the Galerkin operator is also full and the coarse levels are thus quite costly to use. We have thus preferred a mesh refinement strategy which provides in this case better numerical results.

The finest problem is a polynomial of degree 20. We compute the different values on a grid of 1 681 points. The results are presented in Figure 4.18. The algorithm reaches the optimal solution within 20 minutes. We observe that both the values of the astigmatism and the error on the reference power are small.

Even if some refinements are possible (for example by choosing a grid with more points, or by defining several polynomials each on a part of the surface and adding some fitting conditions between each parts), these solutions respect the quality criteria fixed by INDO.

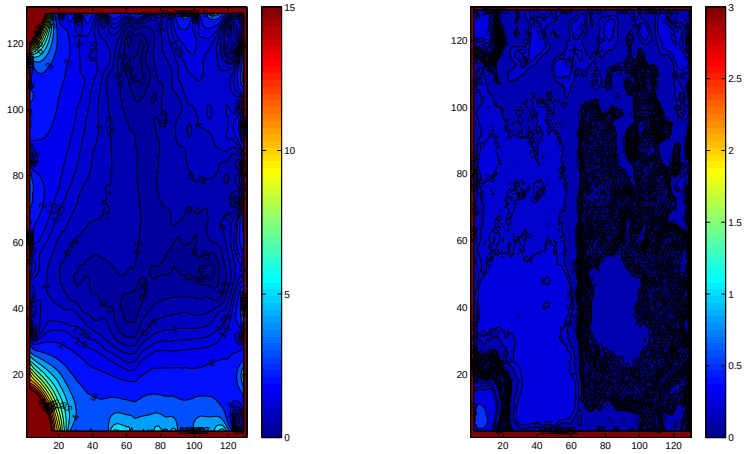

Figure 4.17: Results of the progressive adaptive lens design with a free boundary. The left picture represents the astigmatism and the right picture represents the error made on the reference power.

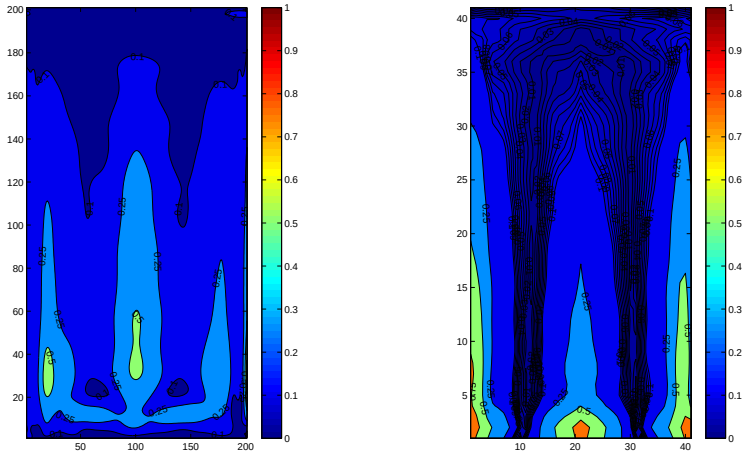

Figure 4.18: Results of the progressive adaptive lens design with a polynomial surface. The left picture represents the astigmatism and the right picture represents the error made on the reference power.

Conclusions and further research perspectives

The purpose of the research work described in this thesis was the design and implementation of algorithms for solving multilevel optimization problems. We have developed algorithms for unconstrained and bound-constrained optimization within a trust-region scheme. In the following paragraphs, we summarize our results and suggest some possible directions for future research.

In the first Chapter of this thesis, after a brief introduction to nonlinear optimization, we have presented the basic trust-region algorithm. We have also explained some refinements of this algorithm to obtain better numerical results. We have especially shown the retrospective trust-region algorithm for which a convergence theory to first-order and second-order critical points is given. This algorithm has also been compared to the classical trust-region algorithm on the unconstrained problems of the CUTEr test set. This algorithm opens the door to larger tests. Indeed, the trust-region algorithm is now widely used and everyone has its own definition of the "basic trust-region" algorithm. But all the algorithms are not always tested on the same problems and on the same computer. It is thus really interesting to test in a same framework all the variants of the trust-region algorithm.

In the second Chapter of this thesis, we have introduced multilevel optimization and described how people have adapted the multigrid philosophy to adapt all the globalization techniques. We have described in more details how we have adapted the trust-region algorithm, first modifying the Moré-Sorensen algorithm which solves exactly the trust-region subproblem and second by using an alternative model in a recursive trust-region framework.

These developments are, we think, just the start of the developments of multilevel optimization and a lot of future works are again possible. We review some of them.

- The first and the most important future development has to be the dealing of general constraints. This could be first done quite fast by modifying an augmented Lagrangian method. However, we think that better results could be obtained by dealing with the constraints using a filter technique modified to take the multilevel structure into account.

- Another interesting development could be inspired by the algebraic techniques. In these methods, the smooth modes of the error are defined as those that are not eliminated fast by the smoothing technique and the transfer operators are constructed accordingly at each iteration. It would be really interesting to construct these methods in optimization where the notion of smooth modes of the error is less clear than in the case of the solution of

linear systems. However, the construction of these transfer operators appears costly and a lot of research has to be done to know how to tackle the problem. Another research possibility which is quite related is the development of adaptive techniques which are already used in multigrid and finite elements methods. In these techniques, one discretizes the domain finely if the error in the associated part of the domain is large. These methods are really interesting since they lead to the solution of an infinite-dimensional problem for a given accuracy with less variables than a standard technique.

- The multigrid methods for linear systems are highly parallelizable. It could be interesting to study the parallelizability of the different multilevel optimization methods.

In the third Chapter of this thesis, we have presented the RMTR package. We have described the algorithmic details of the methods and how they are implemented. And finally, in the fourth Chapter, we have presented the library of multilevel test problems we have constructed and the numerical results obtained on these problems with the package. We have also presented the numerical results obtained with the package on a progressive adaptive lens design industrial problem. It would be really interesting to continue the development of the problems library and to multiply contacts with industry.

In conclusion, this thesis follows the ever-growing needs of industry. Applications become more and more complex and the optimization community has to develop methods to tackle these ones by tacking their underlying nature into account.

Summary of contributions

Our contribution is the design of efficient numerical methods for the solution of infinite-dimensional optimization problems. We have also developed a library of test problems for testing the multilevel algorithms and implemented a software around the RMTR algorithm. We summarize our contributions

- The retrospective algorithm for solving unconstrained nonlinear optimization problems (see Section 1.5.5.1), its convergence analysis (see Section 1.5.5.2) and numerical experiments (see Section 1.5.5.3) have been first presented in Bastin, Malmedy, Mouffe, Toint and Tomanos (2009).

- The application of the multilevel philosophy for the exact solution of the trust-region subproblem has been developed by modifying the Moré-Sorensen method (see Section 2.4). The proof of convergence and some numerical experiences are presented in Toint, Tomanos and Weber-Mendonca (2009).

- Extensive numerical experiences with the RMTR software we have implemented (Chapter 4) together with the description of the library of multilevel problems (Appendix B) are the subject of the paper Gratton et al. (2009).

- The RMTR software has been included in the Galahad library of nonlinear optimization solvers.

- Our progressive lens industrial application (Section 4.4) has been presented in a paper Toint and Tomanos (2009).

Main notations and abbreviations

General

$\mathbb{N}$	set of nonnegative integers
$\mathbb{R}$	set of real numbers
$\mathbb{R}^n$	real n-dimensional Euclidean space
$\|\cdot\|$	absolute value of a scalar
$\|\|\cdot\|\|$	vector norm (Euclidean unless otherwise specified)
$\sharp\mathcal{S}$	cardinality of the set $\mathcal{S}$
$\nabla_x f(x)$	gradient of f
$\nabla^2_{xx} f(x)$	Hessian matrix of f
$P[\cdot]$	projection operator
ε_M	machine precision

Mathematical programming

$\mathcal{E}$	set of equality constraints
$\mathcal{I}$	set of inequality constraints
x^*	optimal solution
Ω	feasible region
$\mathcal{N}$	neighbourhood
$\mathcal{A}(x)$	active set at x
$\mathcal{B}(x^*)$	binding set at the solution
$\mathcal{B}_s(x^*)$	strictly binding set at the solution
$A \otimes B$	Kronecker product of matrix A and B

Algorithms

x_k	kth iterate (vector)
$\mathcal{B}_k$	trust region at iteration k
Δ_k	trust-region radius at iteration k
g_k	gradient of the objective function at x_k
$\bar{g}_k$	"projected" gradient of the objective function at x_k
H_k	symmetric approximation to the objective Hessian at x_k
s_k	step at iteration k
x_k^C	(generalized) Cauchy point
$m_k(\cdot)$	model of f at the kth iteration
ρ_k	ratio of actual to predicted decrease
$\tilde{\rho}_k$	retrospective ratio of actual to predicted decrease
$\mathcal{S}$	set of indices of successful iterations
λ	Lagrangian multiplier associated to the trust-region constraint
ϵ_g	Accuracy threshold
$f_{\min}$	Minimal value allowed for the objective function
$k_{\max}$	Maximal number of iterations allowed

Main mathematical notations

Multigrid methods for linear systems

Ω	Considered domain
Γ	Boundary of the domain
$\mathcal{L}_\Omega$	Differential operator on the domain
$\mathcal{L}_\Gamma$	Differential operator on the boundary
f_Ω	Right-hand-side defined on the domain
f_Γ	Right-hand-side defined on the boundary
G^h	Grid with a meshsize of h
$\cdot^h$	Quantity defined on a grid of meshsize h
I_{2h}^h	Prolongation operator
I_h^{2h}	Restriction operator

Multilevel Optimization

i	Level
P_i	Prolongation operator
R_i	Restriction operator
f_i	Objective function on the i-th level
m_i	Taylor model
h_{i-1}	Coarse model

Multilevel notations

AF	All on Finest
BTR	Basic Trust Region
CG	Conjugate Gradient
FM	Full Multilevel
FMG	Full Multigrid
MF	Multilevel on Finest
MMS	Multilevel Moré-Sorensen
MR	Mesh Refinement
MS	Moré-Sorensen
NLP	Nonlinear Programming
PAL	Progressive Adaptive Lens
PTCG	Projected Truncated Conjugate Gradient
RMTR	Recursive Multilevel Trust Region
SCM	Sequential Coordinate Minimization
TCG	Truncated Conjugate Gradient

Main abbreviations

BRATU	Bratu problem
DEPT	An elastic-plastic torsion problem
DNT	A Dirichlet-to-Neumann transfer problem
DODC	An optimal design with composite materials
DPJB	Pressure distribution in a journal bearing
DSSC	Steady-state combustion problem
IGNISC	A solid-ignition problem
MEMBR	A membrane problem
MINS-BC	Minimal surface problem with bound constraints
MINS-DMSA	Enneper problem
MINS-OB	Minimal surface problem with oscillatory boundary
MINS-SB	Minimal surface problem with smooth boundary
MOREBV	A nonlinear boundary value problem
NCCO	Optimal control problem with oscillatory boundary
NCCS	Optimal control problem with smooth boundary
P2D	Poisson two-dimensional problem
P3D	Poisson three-dimensional problem

Multilevel problems

Appendix

Appendix A

Complete numerical results of the retrospective algorithm

Here is the set of results from our tests. For each problem, we report its number of variables (n), the number of iterations ($iter$), the number of gradient evaluations ($\#g$) and the best objective function value found (f). The symbol $>$ indicates that the iteration limit (fixed at 100 000) was exceeded. The columns "LS" contains a star for least-squares problems.

			Moré-Sorensen						Steihaug-Toint					
			BTR			RTR			BTR			RTR		
Name	LS	n	iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
AKIVA		2	6	7	6.1660e+00	6	7	6.1660e+00	8	9	6.1660e+00	8	9	6.1660e+00
ALLINITU		4	7	8	5.7444e+00	7	8	5.7444e+00	5	6	5.7444e+00	5	6	5.7444e+00
ARGLINA	*	200	5	6	2.0000e+02	5	6	2.0000e+02	5	6	2.0000e+02	5	6	2.0000e+02
ARWHEAD		100	5	6	6.5947e−14	5	6	6.5947e−14	5	6	0.0000e+00	5	6	0.0000e+00
BARD	*	3	9	9	8.2149e−03	9	9	8.2149e−03	13	13	8.2149e−03	13	13	8.2149e−03
BDQRTIC	*	100	10	11	3.7877e+02	10	11	3.7877e+02	13	14	3.7877e+02	13	14	3.7877e+02
BEALE	*	2	9	9	1.9232e−16	8	8	4.5813e−14	7	8	7.3194e−12	7	8	7.3194e−12
BIGGS6	*	6	6094	4585	2.4268e−01	6021	4685	2.4268e−01	149	135	8.9467e−09	149	138	1.6487e−07
BOX3	*	3	7	8	1.5192e−11	7	8	1.5192e−11	8	9	2.3841e−15	8	9	2.3841e−15
BRKMCC		2	2	3	1.6904e−01	2	3	1.6904e−01	3	4	1.6904e−01	3	4	1.6904e−01
BROWNAL	*	200	24	20	5.3204e−23	32	27	1.2675e−15	5	6	1.4731e−09	5	6	1.4731e−09
BROWNBS	*	2	29	29	0.0000e+00	29	29	0.0000e+00	51	52	0.0000e+00	55	56	0.0000e+00
BROWNDEN	*	4	10	11	8.5822e+04	10	11	8.5822e+04	11	12	8.5822e+04	11	12	8.5822e+04
BROYDN7D		100	24	21	3.9739e+01	23	21	3.9771e+01	35	31	3.9660e+01	31	27	3.9660e+01
BRYBND	*	100	17	13	2.0687e−28	12	12	1.4121e−23	11	12	2.8661e−17	11	12	2.8661e−17
CHAINWOO	*	100	53	44	1.0000e+00	50	45	1.0000e+00	300	228	5.5035e+01	162	141	3.2191e+01
CHNROSNB	*	50	57	48	1.8917e−13	54	50	2.4837e−21	78	61	6.7337e−14	64	59	2.2256e−15
CLIFF		2	27	28	1.9979e−01	27	28	1.9979e−01	30	31	1.9979e−01	30	31	1.9979e−01
COSINE		100	6	7	-9.9000e+01	6	7	-9.9000e+01	10	10	-9.9000e+01	10	10	-9.9000e+01
CRAGGLVY		202	15	16	6.6741e+01	15	16	6.6741e+01	16	17	6.6741e+01	16	17	6.6741e+01
CUBE	*	2	37	31	9.3052e−12	35	31	1.9212e−15	44	38	1.2297e−12	42	37	1.7564e−13
CURLY10	*	50	9	10	-5.0158e+03	9	10	-5.0158e+03	18	18	-5.0158e+03	18	18	-5.0158e+03
CURLY20	*	50	8	9	-5.0158e+03	8	9	-5.0158e+03	18	18	-5.0158e+03	18	18	-5.0158e+03
CURLY30	*	50	13	13	-5.0158e+03	13	13	-5.0158e+03	17	16	-5.0158e+03	20	19	-5.0158e+03
DECONVU	*	61	25	19	1.9290e−10	19	16	1.7251e−08	22	19	3.9035e−08	22	20	3.9966e−08
DENSCHNA		2	5	6	2.2139e−12	5	6	2.2139e−12	5	6	1.2000e−15	5	6	1.2000e−15
DENSCHNB	*	2	4	5	3.3850e−16	4	5	3.3850e−16	6	7	7.9948e−14	6	7	7.9948e−14
DENSCHNC	*	2	10	11	2.1777e−20	10	11	2.1777e−20	9	10	1.8423e−13	9	10	1.8423e−13
DENSCHND	*	3	37	33	1.1392e−08	38	34	1.1392e−08	30	31	1.3753e−08	30	31	1.3753e−08
DENSCHNE	*	3	9	10	8.7102e−19	9	10	8.7102e−19	16	16	4.4587e−19	15	16	7.3809e−13
DENSCHNF	*	2	6	7	6.5132e−22	6	7	6.5132e−22	6	7	6.5132e−22	6	7	6.5132e−22

			Moré-Sorensen						Steihaug-Toint					
			BTR			RTR			BTR			RTR		
Name	LS	n	iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
DIXMAANA		150	7	8	1.0000e+00	7	8	1.0000e+00	9	10	1.0000e+00	9	10	1.0000e+00
DIXMAANB		150	11	11	1.0000e+00	11	11	1.0000e+00	9	10	1.0000e+00	9	10	1.0000e+00
DIXMAANC		150	11	11	1.0000e+00	11	11	1.0000e+00	10	11	1.0000e+00	10	11	1.0000e+00
DIXMAAND		150	14	13	1.0000e+00	14	13	1.0000e+00	11	12	1.0000e+00	11	12	1.0000e+00
DIXMAANE		150	10	10	1.0000e+00	11	11	1.0000e+00	11	11	1.0000e+00	11	12	1.0000e+00
DIXMAANF		150	15	14	1.0000e+00	14	13	1.0000e+00	12	13	1.0000e+00	12	13	1.0000e+00
DIXMAANG		150	15	14	1.0000e+00	15	14	1.0000e+00	13	14	1.0000e+00	13	14	1.0000e+00
DIXMAANH		150	18	16	1.0000e+00	19	17	1.0000e+00	14	15	1.0000e+00	14	15	1.0000e+00
DIXMAANI		150	14	14	1.0000e+00	16	16	1.0000e+00	13	14	1.0000e+00	13	14	1.0000e+00
DIXMAANJ		150	25	21	1.0000e+00	18	16	1.0000e+00	18	17	1.0000e+00	19	18	1.0000e+00
DIXMAANK		150	23	20	1.0000e+00	19	17	1.0000e+00	22	20	1.0000e+00	20	19	1.0000e+00
DIXMAANL		150	23	20	1.0000e+00	25	22	1.0000e+00	15	16	1.0000e+00	15	16	1.0000e+00
DIXON3DQ		100	4	5	1.1710e−29	4	5	1.1710e−29	8	9	0.0000e+00	8	9	0.0000e+00
DJTL		2	105	71	-8.9515e+03	104	74	-8.9515e+03	231	161	-8.9515e+03	253	183	-8.9515e+03
DQDRTIC		100	5	6	2.3990e−28	5	6	2.3990e−28	9	10	1.7453e−17	9	10	1.7453e−17
DQRTIC		100	29	30	2.8059e−08	29	30	2.8059e−08	29	30	3.5899e−08	29	30	3.5899e−08
EDENSCH		100	19	18	6.0328e+02	20	19	6.0328e+02	17	18	6.0328e+02	17	18	6.0328e+02
EG2		100	3	4	-9.8947e+01	3	4	-9.8947e+01	3	4	-9.8947e+01	3	4	-9.8947e+01
EIGENALS	*	110	20	21	5.0766e−21	20	20	1.1113e−12	23	23	1.0531e−12	23	23	8.3333e−13
EIGENBLS	*	110	134	107	4.2412e−15	69	63	3.1853e−17	164	142	3.7937e−13	167	153	1.3427e−12
ENGVAL1		100	9	10	1.0909e+02	9	10	1.0909e+02	11	12	1.0909e+02	11	12	1.0909e+02
ENGVAL2	*	3	13	14	9.7152e−17	13	14	9.7152e−17	24	24	5.2007e−15	24	24	1.1952e−15
ERRINROS	*	50	56	48	3.9904e+01	52	47	3.9904e+01	85	79	3.9904e+01	75	72	3.9904e+01
EXPFIT	*	2	7	6	2.4051e−01	7	6	2.4051e−01	13	12	2.4051e−01	16	14	2.4051e−01
EXTROSNB	*	100	1281	1182	1.8373e−08	487	468	3.1722e−07	566	516	1.5784e−06	643	624	7.1530e−07
FLETCBV2		100	2	3	-5.1401e−01	2	3	-5.1401e−01	3	4	-5.1401e−01	3	4	-5.1401e−01
FLETCBV3		50	>	>	-3.5073e+02	>	>	-3.3920e+02	30878	30541	-1.3860e+03	>	>	-1.0286e+03
FLETCHBV		10	460	453	-2.1502e+06	1203	1151	-2.0203e+06	127	118	-2.3674e+06	257	257	-2.1109e+06
FLETCHCR		100	231	200	1.7096e−19	164	162	2.6432e−19	347	264	1.2049e−14	194	180	7.8105e−18
FMINSRF2		121	35	31	1.0000e+00	30	25	1.0000e+00	95	91	1.0000e+00	70	60	1.0000e+00
FMINSURF		121	32	27	1.0000e+00	23	19	1.0000e+00	102	98	1.0000e+00	70	59	1.0000e+00

			Moré-Sorensen						Steihaug-Toint					
			BTR			RTR			BTR			RTR		
Name	LS	n	iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
FREUROTH	*	100	9	10	1.1965e+04	9	10	1.1965e+04	14	15	1.1965e+04	14	15	1.1965e+04
GENHUMPS	*	10	10402	9802	3.7851e−12	11624	10931	4.3255e−13	5083	4434	6.3997e−13	7075	6449	2.7198e−14
GENROSE	*	100	107	88	1.0000e+00	90	83	1.0000e+00	130	116	1.0000e+00	123	113	1.0000e+00
GENROSEB		500	460	369	1.0000e+00	327	325	1.0000e+00	585	505	1.0000e+00	498	473	1.0000e+00
GROWTHLS	*	3	96	78	1.0040e+00	79	72	1.0040e+00	183	172	1.0040e+00	171	163	1.0040e+00
GULF	*	3	30	28	1.7991e−17	32	30	3.6188e−14	40	38	3.4547e−13	44	43	3.2415e−09
HAIRY		2	64	57	2.0000e+01	116	107	2.0000e+01	96	84	2.0000e+01	91	86	2.0000e+01
HATFLDD	*	3	20	20	6.6151e−08	20	20	6.6151e−08	18	18	6.6937e−08	18	18	6.6937e−08
HATFLDE	*	3	21	21	5.1204e−07	20	20	5.1204e−07	17	17	5.1204e−07	17	17	5.1204e−07
HEART6LS	*	6	667	642	4.4113e−26	1039	1019	2.1192e−24	1528	1498	7.2910e−13	1593	1583	1.5966e−12
HEART8LS	*	8	112	95	4.6362e−17	102	88	1.7507e−13	152	143	2.0524e−20	159	154	3.8145e−14
HELIX	*	3	11	11	5.6587e−23	8	8	4.9599e−13	20	19	7.7395e−15	15	14	1.8475e−15
HIELOW		3	11	10	8.7417e+02	8	8	8.7417e+02	13	12	8.7417e+02	12	11	8.7417e+02
HILBERTA		2	3	4	2.0543e−33	3	4	2.0543e−33	3	4	1.8551e−30	3	4	1.8551e−30
HILBERTB		10	3	4	1.8835e−29	3	4	1.8835e−29	7	8	2.2225e−14	7	8	2.2225e−14
HIMMELBB		2	10	9	5.1740e−16	10	8	1.2423e−20	19	19	1.7548e−11	19	19	1.7548e−11
HIMMELBF	*	4	276	274	3.1857e+02	94	92	3.1857e+02	358	356	3.1857e+02	353	315	3.1857e+02
HIMMELBG		2	5	6	9.0327e−12	5	6	9.0327e−12	7	7	1.7308e−15	7	7	1.7308e−15
HIMMELBH		2	4	5	-1.0000e+00	4	5	-1.0000e+00	4	5	-1.0000e+00	4	5	-1.0000e+00
HUMPS	*	2	2690	2503	1.0977e−12	6856	6604	2.4027e−13	2606	2243	6.0915e−14	6265	6038	6.5371e−11
JENSMP		2	9	10	1.2436e+02	9	10	1.2436e+02	9	10	1.2436e+02	9	10	1.2436e+02
KOWOSB	*	4	11	10	3.0780e−04	11	10	3.0780e−04	12	12	3.0780e−04	12	11	3.0780e−04
LIARWHD	*	100	12	13	5.5677e−14	12	13	5.5677e−14	14	15	2.4677e−15	14	15	2.4677e−15
LOGHAIRY		2	2734	2676	1.8232e−01	9091	8167	1.8232e−01	4871	4132	5.1277e+00	7612	6953	1.8232e−01
MANCINO	*	100	14	15	1.5058e−21	16	16	4.0607e−19	20	21	1.4487e−21	20	21	1.4487e−21
MARATOSB		2	699	673	-1.0000e+00	680	667	-1.0000e+00	1882	1726	-1.0000e+00	1547	1493	-1.0000e+00
MEXHAT		2	32	30	-4.0010e−02	31	30	-4.0010e−02	19	20	-4.0010e−02	19	20	-4.0010e−02
MEYER3	*	3	481	441	8.7946e+01	416	381	8.7946e+01	686	680	8.8511e+01	693	688	8.8186e+01
MODBEALE		200	10	11	7.8240e−21	10	11	7.8240e−21	14	15	3.1114e−15	14	15	3.1114e−15
MOREBV	*	100	1	2	7.8870e−10	1	2	7.8870e−10	138	139	2.1401e−07	138	139	2.1401e−07
MSQRTALS	*	100	20	18	2.6765e−17	19	17	7.4695e−10	20	19	4.0318e−11	20	19	4.0318e−11

			Moré-Sorensen						Steihaug-Toint					
			BTR			RTR			BTR			RTR		
Name	LS	n	iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
MSQRTBLS	*	100	16	14	1.8855e−17	16	14	9.4179e−14	21	20	4.1329e−14	21	20	4.1329e−14
NONCVXU2		100	53	47	2.3183e+02	49	41	2.3241e+02	45	40	2.3241e+02	41	34	2.3241e+02
NONCVXUN		100	42	38	2.3168e+02	41	36	2.3285e+02	44	40	2.3168e+02	41	34	2.3227e+02
NONDIA	*	100	6	7	1.4948e−18	6	7	1.4948e−18	10	11	6.5982e−15	10	11	6.5982e−15
NONDQUAR		100	15	16	2.6991e−09	15	16	2.6991e−09	110	84	2.1978e−06	97	86	1.9731e−06
OSBORNEA	*	5	37	32	5.4649e−05	30	27	5.4649e−05	64	59	5.4718e−05	82	79	5.4649e−05
OSBORNEB	*	11	21	19	4.0138e−02	21	19	4.0138e−02	22	22	4.0138e−02	22	22	4.0138e−02
OSCIPATH		8	2035	1734	1.7473e−05	2015	1804	1.4813e−05	3020	2625	3.3662e−05	2670	2488	4.3935e−05
PALMER1C		8	7	8	9.7605e−02	7	8	9.7605e−02	>	>	9.7653e−02	>	>	9.7653e−02
PALMER1D		7	7	8	6.5267e−01	7	8	6.5267e−01	23	24	6.5267e−01	23	24	6.5267e−01
PALMER2C		8	6	7	1.4369e−02	6	7	1.4369e−02	3161	3162	1.4370e−02	3161	3162	1.4370e−02
PALMER3C		8	6	7	1.9538e−02	6	7	1.9538e−02	1784	1785	1.9539e−02	1784	1785	1.9539e−02
PALMER4C		8	7	8	5.0311e−02	7	8	5.0311e−02	1538	1539	5.0312e−02	1538	1539	5.0312e−02
PALMER5C	*	6	5	6	2.1281e+00	5	6	2.1281e+00	9	10	2.1281e+00	9	10	2.1281e+00
PALMER6C	*	8	7	8	1.6387e−02	7	8	1.6387e−02	165	166	1.6388e−02	165	166	1.6388e−02
PALMER7C	*	8	9	10	6.0199e−01	9	10	6.0199e−01	6810	5734	6.0199e−01	4456	3946	6.0199e−01
PALMER8C	*	8	8	9	1.5977e−01	8	9	1.5977e−01	197	198	1.5977e−01	197	198	1.5977e−01
PENALTY1	*	100	45	44	9.0249e−04	45	44	9.0249e−04	44	41	9.0260e−04	48	44	9.0249e−04
PENALTY2	*	100	19	20	9.7096e+04	19	20	9.7096e+04	19	20	9.7096e+04	19	20	9.7096e+04
PFIT1LS	*	3	325	287	1.5734e−16	294	280	3.0857e−15	365	350	4.8505e−07	384	379	4.3509e−07
PFIT2LS	*	3	114	98	3.6218e−15	90	84	3.4229e−20	133	128	1.9620e−08	161	158	7.5351e−09
PFIT3LS	*	3	144	125	4.4639e−19	126	116	3.6432e−14	222	211	1.2519e−08	226	221	2.4788e−09
PFIT4LS	*	3	241	218	3.4144e−20	232	223	8.8142e−23	401	390	6.1391e−10	495	491	7.1420e−10
POWELLSG		4	15	16	4.6333e−09	15	16	4.6333e−09	15	16	1.2731e−08	15	16	1.2731e−08
POWER		100	24	25	1.1818e−09	24	25	1.1818e−09	25	26	1.6694e−09	25	26	1.6694e−09
QUARTC		100	29	30	2.8059e−08	29	30	2.8059e−08	29	30	3.5899e−08	29	30	3.5899e−08
ROSENBR	*	2	30	26	7.1488e−15	28	26	6.0210e−13	34	30	2.8234e−14	34	31	5.7977e−11
S308	*	2	13	12	7.7320e−01	13	12	7.7320e−01	9	10	7.7320e−01	9	10	7.7320e−01
SBRYBND	*	100	46	37	2.5620e−22	46	37	9.1262e−15	>	>	2.6525e+01	>	>	2.5463e+01
SCHMVETT		100	4	5	-2.9400e+02	4	5	-2.9400e+02	6	7	-2.9400e+02	6	7	-2.9400e+02
SCOSINE		100	>	>	-9.8840e+01	97	90	-9.9000e+01	>	>	-9.7311e+01	>	>	-9.3382e+01

			MORÉ-SORENSEN						STEIHAUG-TOINT					
			BTR			RTR			BTR			RTR		
Name	LS	n	iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
SCURLY10	*	100	39	35	-1.0032e+04	46	42	-1.0032e+04	>	>	-1.0013e+04	>	>	-1.0013e+04
SCURLY20	*	100	34	30	-1.0032e+04	37	33	-1.0032e+04	>	>	-1.0032e+04	>	>	-1.0032e+04
SCURLY30	*	100	35	31	-1.0032e+04	35	31	-1.0032e+04	>	>	-1.0022e+04	>	>	-1.0021e+04
SENSORS	*	100	21	21	-1.9668e+03	24	23	-1.9668e+03	20	20	-2.0250e+03	24	22	-2.0250e+03
SINEVAL	*	2	53	46	1.9744e−25	58	52	3.3812e−36	107	93	3.6189e−18	80	73	1.4447e−21
SINQUAD		100	9	10	-4.0056e+03	9	10	-4.0056e+03	14	14	-4.0056e+03	11	12	-4.0056e+03
SISSER		2	12	13	1.0658e−08	12	13	1.0658e−08	12	13	1.2144e−08	12	13	1.2144e−08
SNAIL		2	61	61	9.3702e−13	59	60	1.2117e−14	72	72	8.6160e−17	62	63	3.6402e−18
SPARSINE		100	37	27	9.3794e−16	30	22	2.8734e−16	10	11	1.7155e−15	10	11	1.7155e−15
SPARSQUR		100	16	17	1.4795e−08	16	17	1.4795e−08	16	17	1.9872e−08	16	17	1.9872e−08
SPMSRTLS	*	100	14	13	1.2592e−13	12	11	6.1356e−12	13	13	4.6661e−14	13	13	4.6661e−14
SROSENBR	*	100	6	7	8.8993e−28	6	7	8.8993e−28	8	9	2.6078e−19	8	9	2.6078e−19
TOINTGOR		50	9	10	1.3739e+03	9	10	1.3739e+03	11	12	1.3739e+03	11	12	1.3739e+03
TOINTGSS		100	17	15	1.0102e+01	13	13	1.0204e+01	12	12	1.0102e+01	12	12	1.0102e+01
TOINTPSP		50	22	20	2.2556e+02	30	28	2.2556e+02	47	38	2.2556e+02	58	50	2.2556e+02
TQUARTIC	*	100	14	13	2.6771e−24	15	13	1.4965e−17	15	15	5.3087e−15	15	15	5.3087e−15
VARDIM		200	29	30	2.9081e−24	29	30	2.9081e−24	29	30	2.0682e−25	29	30	2.0682e−25
VAREIGVL	*	50	15	13	4.7122e−09	16	14	1.3553e−10	13	14	2.2712e−10	13	14	2.2712e−10
VIBRBEAM	*	8	49	39	1.7489e+00	51	40	1.7489e+00	668	669	1.5645e−01	960	956	1.5645e−01
WATSON	*	12	14	14	8.1544e−07	13	13	3.9067e−08	12	13	1.5973e−07	12	13	1.5973e−07
WOODS	*	4	52	44	4.6408e−15	53	47	5.1563e−17	69	59	2.0670e−13	60	54	3.8275e−17
YFITU	*	3	54	48	6.6863e−13	50	46	6.6700e−13	85	77	2.2960e−08	79	75	1.0173e−08

Appendix B

Test problems

We have built a suite of test problems as extensive as we could, from a variety of sources. We have kept the problems already discussed in Gratton et al. (2006) and have also used Lewis and Nash (2005) and the Minpack-2 collection (Averick and Moré, 1991). In what follows, we denote by S_2 and S_3 respectively the unit square and cube

$$S_2 = [0,1] \times [0,1] = \{(x,y), 0 \leq x \leq 1, 0 \leq y \leq 1\}$$

and

$$S_3 = [0,1] \times [0,1] \times [0,1] = \{(x,y,z), 0 \leq x \leq 1, 0 \leq y \leq 1, 0 \leq z \leq 1\}.$$

We also denote by $\mathcal{H}^1(\mathcal{D})$ the Hilbert space of all functions with compact support in the domain $\mathcal{D}$ such that v and $||\nabla v||^2$ belong to $\mathcal{L}^2(\mathcal{D})$, and by $\mathcal{H}^1_0(\mathcal{D})$ its subspace consisting of all function vanishing on the domain's boundary. For all problems, the starting value of the unknown function is chosen to be equal to one (at the finest level).

B.1 DNT: a Dirichlet-to-Neumann transfer problem

Let S be the square $[0,\pi] \times [0,\pi]$ and let Γ be its lower edge defined by $\{(x,y), 0 \leq x \leq \pi, y = 0\}$. The Dirichlet-to-Neumann transfer problem (Lewis and Nash, 2005) consists of finding the function $a(x)$ defined on $[0,\pi]$, that minimizes

$$\int_0^\pi \left(\frac{\partial u}{\partial y}(x,0) - f(x) \right)^2,$$

where $u(x,y)$ is the solution of the boundary value problem

$$\begin{array}{lcll} \Delta u & = & 0 & \text{in } S, \\ u(x,y) & = & a(x) & \text{on } \Gamma, \\ u(x,y) & = & 0 & \text{on } \partial S \backslash \Gamma, \end{array}$$

and Δ is the Laplacian operator. The problem is a 1D minimization problem, but the computations of the objective function, gradient and Hessian involve a partial differential equation in 2D. To introduce oscillatory components in the solution, we define $f(x) = \sum_{i=1}^{15} \sin(i\,x) + \sin(40\,x)$. The discretization of the problem is performed by finite differences with the same grid spacing in the two directions. The discretized problem is a linear least-squares problem.

B.2 P2D and P3D: two quadratic examples

We consider here the two-dimensional Poisson model problem P2D for multigrid solvers defined in S_2

$$\begin{aligned} -\Delta u(x) &= f(x) \text{ in } S_2, \\ u(x) &= 0 \text{ on } \partial S_2, \end{aligned}$$

where $f(x)$ is such that the analytical solution to this problem is $u(x) = 2x_2(1-x_2)+2x_1(1-x_1)$. This problem is discretized using a 5-point finite-difference scheme. We consider the variational formulation of this problem, given by

$$\min_{x \in \mathbb{R}^{n_r}} \tfrac{1}{2} x^T A x - x^T b, \tag{B.1}$$

which is obviously equivalent to the linear system $Ax = b$, where A and b are the discretizations of the Laplacian and the right-hand side f, respectively. The main purpose of this example is to illustrate that our multilevel algorithm exhibits performances similar to traditional linear multigrid solvers on a quadratic model problem.

Problem P3D is a more nonlinear 3D version of P2D. We consider the differential equation

$$\begin{aligned} -(1+\sin^2(3\pi x_1))\Delta u(x) &= f(x) \quad \text{in } S_3, \\ u(x) &= 0 \quad \text{on } \partial S_3. \end{aligned}$$

The right-hand side $f(x)$ is chosen such that $u(x) = x_1(1-x_1)x_2(1-x_2)x_3(1-x_3)$ is the desired solution. The Laplacian is discretized using the standard 7-point finite-difference approximation on a uniform 3D mesh. As for P2D, the solution algorithms are applied to the variational formulation (B.1).

B.3 MINS-SB, MINS-OB, MINS-BC and MINS-DMSA: four minimum surface problems

The domain of calculus of variation consists of finding stationary values v of integrals of the form $\int_a^b f(v, \dot{v}, x)\, dx$, where $\dot{v}$ is the first-order derivative of v. The multilevel trust-region algorithm can be applied to discretized versions of problems of this type. As representative of these, we consider several variants of the minimum surface problem

$$\min_{v \in \mathcal{K}} \int_{S_2} \sqrt{1 + \|\nabla_x v\|_2^2},$$

where $\mathcal{K} = \{v \in H^1(S_2) \mid v(x) = v_0(x) \text{ on } \partial S_2\}$. This convex problem is discretized using a finite-element basis defined using a uniform triangulation of S_2, with the same grid spacing, h, along the two coordinate directions. The basis functions are the classical P1 functions which are linear on each triangle and take the value 0 or 1 at each vertex. The boundary condition

$v_0(x)$ is chosen as

$$v_0(x) = \begin{cases} f(x_1), & x_2 = 0, \quad 0 \le x_1 \le 1, \\ 0, & x_1 = 0, \quad 0 \le x_2 \le 1, \\ f(x_1), & x_2 = 1, \quad 0 \le x_1 \le 1, \\ 0, & x_1 = 1, \quad 0 \le x_2 \le 1, \end{cases}$$

where $f(x_1) = x_1(1 - x_1)$ (for MINS-SB) or $f(x_1) = \sin(4\pi x_1) + \frac{1}{10}\sin(120\pi x_1)$ (for MINS-OB). To define problem MINS-BC, we introduce, in MINS-SB, the following lower bound constraint:

$$v(x) \ge \sqrt{2} \qquad \text{whenever} \quad \frac{4}{9} \le x_1, x_2 \le \frac{5}{9},$$

thereby creating an obstacle problem where the surface is constrained in the middle of the domain. The fourth variant of the minimum surface problem, MINS-DMSA, is the Enneper problem proposed in Minpack-2, where the domain is now given by $\mathcal{D} = (-\frac{1}{2}, \frac{1}{2}) \times (-\frac{1}{2}, \frac{1}{2})$. The boundary condition is chosen on $\partial\mathcal{D}$ as

$$v_{\mathcal{D}}(x) = u^2 - v^2,$$

where u and v are the unique solutions to the equations

$$x_1 = u + uv^2 - \frac{1}{3}u^3, \quad x_2 = -v - u^2 v + \frac{1}{3}v^3.$$

B.4 MEMBR: a membrane problem

We consider the problem suggested by Domorádová and Dostál (2007) given by

$$\min_{u \in \mathcal{K}} \int_{S_2} \Big(||\nabla u(x)||_2^2 + u(x) \Big)$$

where the boundary of S_2 is composed of three parts: $\Gamma_u = \{0\} \times [0, 1]$, $\Gamma_l = \{1\} \times [0, 1]$ and $\Gamma_f = [0, 1] \times \{0, 1\}$ and where $\mathcal{K} = \{u \in \mathcal{H}^1(S_2) \mid u(x) = 0 \ \text{ on } \ \Gamma_u \ \text{ and } \ l \le u(x) \ \text{ on } \ \Gamma_l\}$. The obstacle l on the boundary Γ_l is defined by the upper part of the circle with the radius one and center $S = (1; 0.5; -1.3)$.

The solution of this problem can be interpreted as the displacement of the membrane under the traction defined by the unit density. The membrane is fixed on Γ_u and is not allowed to penetrate the obstacle on Γ_l. We discretized the problem by piecewise linear finite elements using a regular triangular grid.

B.5 IGNISC, DSSC and BRATU: three combustion/Bratu problems

We first consider the following optimal-control problem (IGNISC), introduced by Borzi and Kunisch (2006), and related to the solid-ignition model:

$$\min_{u \in \mathcal{H}_0^1(S_2)} \left[\int_{S_2} \Big(u(x) - z \Big)^2 + \frac{\beta}{2} \int_{S_2} \Big(e^{u(x)} - e^z \Big)^2 + \frac{\nu}{2} \int_{S_2} \| \Delta u(x) - \delta e^{u(x)} \|_2^2 \right].$$

For the numerical tests, we chose $\nu = 10^{-5}$, $\delta = 6.8$, $\beta = 6.8$ and $z = \frac{1}{\pi^2}$.

The second problem of this type is the steady-state combustion problem DSSC of Minpack 2, stated as the infinite-dimensional optimization problem

$$\min_{u \in \mathcal{H}_0^1(S_2)} \int_{S_2} \left(\tfrac{1}{2} ||\nabla u(x)||_2^2 - \lambda e^{u(x)} \right)$$

with $\lambda = 5$. This problem is the variational formulation of the boundary value problem

$$\begin{aligned} -\Delta u(x) &= \lambda e^{u(x)}, \quad x \in S_2, \\ u(x) &= 0, \qquad\quad x \in \partial S_2. \end{aligned}$$

The third variant is a simple least-squares formulation of the same problem, where we solve

$$\min_{u \in \mathcal{H}_0^1(S_2)} \int_{S_2} \|\Delta u(x) + \lambda e^{u(x)}\|_2^2,$$

with $\lambda = 6.8$. For all these convex problems, we use standard 5-point finite differences on a uniform grid.

B.6 NCCS and NCCO: two nonconvex optimal control problems

We introduce the nonlinear least-squares problem

$$\min_{u,v \in \mathcal{H}_0^1(S_2)} \left[\int_{S_2} (u(x) - u_0(x))^2 + \int_{S_2} (v(x) - v_0(x))^2 + \int_{S_2} \|\Delta u(x) - v(x)u(x) + f_0(x)\|_2^2 \right].$$

We distinguish two variants: the first with relatively smooth target functions and the second with more oscillatory ones. Theses functions $v_0(x)$ and $u_0(x)$ are defined on S_2 by

$$\begin{aligned} v_0(x) = u_0(x) &= \sin(6\pi x_1)\sin(2\pi x_2) \qquad \text{(for NCCS)} \\ v_0(x) = u_0(x) &= \sin(128\pi x_1)\sin(32\pi x_2) \qquad \text{(for NCCO)} \quad . \end{aligned}$$

The function $f_0(x)$ is such that $-\Delta u_0(x) + v_0(x)u_0(x) = f_0(x)$ on S_2. This problem corresponds to a penalized version of a constrained optimal control problem, and is discretized using finite differences. The nonconvexity of the resulting discretized fine-grid problem has been assessed by a direct eigenvalue computation on the Hessian of the problem.

B.7 DPJB: pressure distribution in a journal bearing

The journal bearing problem arises in the determination of the pressure distribution in a thin film of lubricant between two circular cylinders. This problem is again proposed by Minpack 2, and is of the form

$$\min_{v \in \mathcal{K}} \tfrac{1}{2} \int_{\mathcal{D}} \left(w_q(x) ||\nabla v(x)||_2^2 - \tfrac{1}{10} w_l(x) v(x) \right),$$

where

$$w_q(x) = (1 + \tfrac{1}{10}\cos x_1)^3 \quad \text{and} \quad w_l(x) = \tfrac{1}{10}\sin x_1$$

for some constant $\epsilon \in (0,1)$ and $\mathcal{D} = (0, 2\pi) \times (0, 20)$. The convex set $\mathcal{K}$ is defined by $\mathcal{K} = \{v \in \mathcal{H}_0^1(\mathcal{D}) \mid v(x) \geq 0 \ \text{ on } \ \mathcal{D}\}$. A finite-element approach of this problem is obtained by minimizing over the space of piecewise linear functions v with values $v_{i,j}$ at $z_{i,j} \in \mathbb{R}^2$, which are the vertices of the regular triangulations of $\mathcal{D}$.

B.8 DEPT: an elastic-plastic torsion problem

The elastic-plastic torsion problem DEPT from Minpack 2 arises from the determination of the stress field on an infinitely long cylindrical bar. The infinite-dimensional version of this problem is of the form

$$\min_{v \in \mathcal{K}} \tfrac{1}{2} \int_{S_2} \Big(||\nabla v(x)||_2^2 - 5v(x) \Big).$$

The convex set $\mathcal{K}$ is defined by $\mathcal{K} = \{v \in \mathcal{H}_0^1(S_2) \mid |v(x)| \leq \text{dist}(x, \partial S_2), \quad \text{on} \quad S_2\}$, where $\text{dist}(., \partial S_2)$ is the distance function to the boundary of S_2. A finite-element approach for this problem is obtained by minimizing over the space of piecewise linear functions v with values $v_{i,j}$ at $z_{i,j} \in \mathbb{R}^2$, which are the vertices of the regular triangulations of S_2.

B.9 DODC: an optimal design with composite materials

The Minpack 2 DODC optimal design problem is defined by

$$\min_{v \in \mathcal{H}_0^1(S_2)} \int_{\mathcal{D}} \Big(\psi_\lambda(||\nabla v(x)||_2) + v(x) \Big),$$

where

$$\psi_\lambda(t) = \begin{cases} \frac{1}{2}\mu_2 t^2, & 0 \leq t \leq t_1, \\ \mu_2 t(t - \frac{1}{2}t_1), & t_1 \leq t \leq t_2, \\ \frac{1}{2}\mu_1(t^2 - t_2^2) + \mu_2 t_1(t_2 - \frac{1}{2}t_1), & t_2 \leq t, \end{cases}$$

with the breakpoints t_1 and t_2 defined by

$$t_1 = \sqrt{2\lambda \frac{\mu_1}{\mu_2}} \quad \text{and} \quad t_2 = \sqrt{2\lambda \frac{\mu_2}{\mu_1}},$$

and we choose $\lambda = 0.008$, $\mu_1 = 1$ and $\mu_2 = 2$. A finite-element approach for this problem is obtained by minimizing over the space of piecewise linear functions v with values $v_{i,j}$ at $z_{i,j} \in \mathbb{R}^2$ which are the vertices of the regular triangulations of S_2.

B.10 MOREBV: a nonlinear boundary value problem

The MOREBV problem is adapted (in infinite dimensions) from Moré et al. (1981) and is described by

$$\min_{u \in \mathcal{H}_0^1(S_2)} \int \|\Delta u(x) - \tfrac{1}{2}[u(x) + \langle e, x\rangle + 1]^3\|_2^2,$$

where e is the vector of all ones. Once again, the problem is discretized by linear finite-elements on regular triangular grids.

Appendix C

Complete numerical results of the multilevel algorithms

We give here the complete numerical results for all test problems and all variants. The columns of the following tables report CPU time (in seconds), the number of matrix-vector products or smoothing cycles and the number of objective function/gradient/Hessian evaluations.

P2D	CPU	Mv prods	Eval f	Eval g	eval H	**DODC**	CPU	Mv prods	Eval f	Eval g	eval H	**DEPT**	CPU	Mv prods	Eval f	Eval g	eval H
FM	26.05	13.52	4.66	3.38	1.33	FM	36.00	218.92	65.98	220.55	0.00	FM	8.58	3.37	1.92	4.43	0.00
MR	569.72	1494.99	2.67	2.67	1.33	MR	184.23	4014.31	38.43	354.44	0.00	MR	95.44	206.38	1.66	4.25	0.00
MF	72.85	52.93	10.00	10.00	1.00	MF	58.58	282.99	93.00	399.00	0.00	MF	69.55	52.93	10.00	18.00	0.00
AF	1122.83	3022.00	4.00	4.00	1.00	AF	894.76	11472.00	493.00	4707.00	0.00	AF	1364.45	3019.00	4.00	12.00	0.00
MINS-SB	CPU	Mv prods	Eval f	Eval g	eval H	**MINS-OB**	CPU	Mv prods	Eval f	Eval g	eval H						
FM	153.92	81.89	26.43	18.62	11.91	FM	27.49	305.67	84.99	61.42	21.33						
MR	3600.00	-	-	-	-	MR	116.73	1807.44	26.93	18.43	25.60						
MF	3600.00	-	-	-	-	MF	70.44	564.15	261.00	185.00	69.00						
AF	3600.00	-	-	-	-	AF	1545.63	5955.00	475.00	388.00	460.00						
NCCS	CPU	Mv prods	Eval f	Eval g	eval H	**MINS-DMSA**	CPU	Mv prods	Eval f	Eval g	eval H						
FM	331.89	69.57	69.77	1100.27	0.00	FM	18.23	88.74	26.89	138.65	0.00						
MR	279.51	1342.26	2.68	57.50	0.00	MR	289.64	2860.34	26.31	242.01	0.00						
MF	3600.00	-	-	-	-	MF	73.41	200.25	137.00	591.00	0.00						
AF	3600.00	-	-	-	-	AF	1196.81	5677.00	428.00	4116.00	0.00						
DPJB	CPU	Mv prods	Eval f	Eval g	eval H	**IGNISC**	CPU	Mv prods	Eval f	Eval g	eval H						
FM	83.61	11.17	16.98	28.98	0.00	FM	398.18	65.60	14.98	13.91	1.34						
MR	247.71	341.66	5.02	17.02	0.00	MR	488.22	1882.86	2.69	2.69	1.36						
MF	1390.02	297.00	297.00	306.00	0.00	MF	398.34	257.11	60.00	46.00	1.00						
AF	3600.00	-	-	-	-	AF	2330.42	11572.00	6.00	6.00	5.00						
MEMBR	CPU	Mv prods	Eval f	Eval g	eval H	**DSSC**	CPU	Mv prods	Eval f	Eval g	eval H						
FM	153.96	76.73	98.43	98.43	1.33	FM	12.11	3.41	1.93	4.85	0.00						
MR	292.43	2103.35	3.00	3.00	1.33	MR	122.32	211.51	1.67	4.68	0.00						
MF	335.25	413.97	203.00	183.00	1.00	MF	1051.56	760.65	165.00	134.00	0.00						
AF	1082.05	7423.00	43.00	43.00	1.00	AF	3183.85	6012.00	6.00	42.00	0.00						
MINS-BC	CPU	Mv prods	Eval f	Eval g	eval H	**BRATU**	CPU	Mv prods	Eval f	Eval g	eval H						
FM	140.02	402.25	551.00	540.88	31.64	FM	10.15	3.68	2.06	1.91	0.33						
MR	524.61	4055.91	413.59	400.60	47.15	MR	91.71	203.00	1.67	1.67	0.33						
MF	161.84	414.09	581.00	560.00	84.00	MF	236.82	184.41	43.00	32.00	1.00						
AF	2706.41	3935.00	1105.00	1001.00	1103.00	AF	2314.11	5458.00	6.00	6.00	4.00						
DNT	CPU	Mv prods	Eval f	Eval g	eval H	**NCCO**	CPU	Mv prods	Eval f	Eval g	eval H						
FM	6.73	33.62	9.33	7.33	1.33	FM	224.20	44.01	35.33	791.37	0.00						
MR	4.58	246.40	2.66	2.66	1.33	MR	3589.62	17993.03	3.33	43.37	0.00						
MF	24.41	131.82	37.00	28.00	1.00	MF	3600.00	-	-	-	-						
AF	5.20	299.00	3.00	3.00	1.00	AF	3600.00	-	-	-	-						
P3D	CPU	Mv prods	Eval f	Eval g	eval H	**MOREBV**	CPU	Mv prods	Eval f	Eval g	eval H						
FM	28.78	39.38	8.92	8.64	1.33	FM	41.73	12.83	4.54	3.60	0.33						
MR	18.33	102.08	2.82	2.74	1.33	MR	3600.00	-	-	-	-						
MF	47.47	64.75	12.00	12.00	1.00	MF	704.88	301.01	55.00	44.00	1.00						
AF	626.07	987.00	257.00	142.00	1.00	AF	3600.00	-	-	-	-						

Appendix D

Specification of the RMTR package

D.1 How to use the package

D.1.1 Matrix storage formats

Because multilevel problems are typically large, the matrices must be stored in sparse format, of which two types are allowed. In the sparse co-ordinate format, only the nonzero entries of the matrices are stored. For example, for the l-th entry of the matrix A, its row index i, column index j and value A_{ij} are stored in the l-th components of the integer arrays `A_row`, `A_col` and real array `A_val`. The order is unimportant, but the total number of entries `A_size` is also required. The user also has the possibility to store the Hessian in a compressed sparse row format. In this format, the matrix A is still stored in the integer arrays `A_row`, `A_col` and real array `A_val`. But the i-th entry of array `A_row` (which length is the number of rows of the matrix plus one) now corresponds to the index of the first component in the other two vectors where an element of the i-th line is stored. The array `A_col` then contains the column index and `A_val` contains the value of the entry. Note that the order of the rows is important for this format.

D.1.2 The GALAHAD symbols

As several of the GALAHAD packages, RMTR makes use of "symbols" that are publicly available in the `GALAHAD_SYMBOLS` module. These symbols are conventional names given to specific integer values, and allow a more natural specification of the various options and parameters of the package. Each symbol provided in the `SYMBOLS` module is of the form `GALAHAD_NAME`, where `NAME` is the name of the symbol. For clarify and conciseness, a symbol will be represented by `GALAHAD_NAME` (in sans-serif upper case font) in what follows. See Section D.2 to see how symbols may be used in the program unit that calls the `RMTR` subroutines.

D.1.3 The derived data types

In addition to the multilevel problem data type, two derived data types are accessible from the package.

D.1.3.1 The multilevel problem type

The derived data type `RMTR_problem_type` is used to hold all the information about the multilevel structure of the problem. It is a double linked chained list where each component of the structure contains all the information about a particular level i and is organized as follows.

`level`	is a scalar variable of type default `INTEGER`, that holds the current level number i.
`nbvar`	is a scalar variable of type default `INTEGER`, that holds the number of variables at the current level i, that is the number of variables used to represent each field of the considered problem.
`x`	is a rank-one allocatable array of dimension `nbvar` and type default `REAL`, that hold the values of the problem variables at the current iterate. The j-th component of `x`, $j = 1, \ldots,$ `nbvar`, contains x_j .
`x_start`	is a rank-one allocatable array of dimension `nbvar` and type default `REAL`, that holds the values of the problem variables at the starting point of the current minimization at the considered level i.
`g`	is a rank-one allocatable array of dimension `nbvar` and type default `REAL`, that holds the values of the objective gradient at the current iterate.
`H_ne`	is a scalar variable of type default `INTEGER`, that holds the number of nonzero elements in the objective Hessian.
`H_val, H_row, H_col`	are rank-one allocatable arrays of type default `REAL`, default `INTEGER` and default `INTEGER`, respectively, that hold the values of the objective Hessian approximation at the current iterate in sparse storage.

`R_ne` is a scalar variable of type default `INTEGER`, that holds the number of nonzero elements in the restriction operator to the coarser level (that is the operator from the current level i to the coarser level $i - 1$).

`R_val, R_row, R_col` are rank-one allocatable arrays of type default `REAL`, default `INTEGER` and default `INTEGER`, respectively, that hold the values of the restriction operator to the coarser level (that is the operator from the current level i to the coarser level $i - 1$) in compressed sparse row storage.

`P_ne` is a scalar variable of type default `INTEGER`, that holds the number of nonzero elements in the prolongation operator to the finer level (that is the operator from the current level i to the finer level $i + 1$).

`P_val, P_row, P_col` are rank-one allocatable arrays of type default `REAL`, default `INTEGER` and default `INTEGER`, respectively, that hold the values of the prolongation operator to the finer level (that is the operator from the current level i to the finer level $i + 1$) in compressed sparse row storage.

`sigma` is a scalar variable of type default `REAL`, that holds the 1-norm of the restriction operator stored in the current structure.

`lower_constraint` is a rank-one allocatable array of dimension `nbvar` and type default `REAL`, that holds the values of the lower bound-constraints on the problem variables at the current level.

`upper_constraint` is a rank-one allocatable array of dimension `nbvar` and type default `REAL`, that holds the values of the upper bound-constraints on the problem variables at the current level.

`nbr_iter` is a scalar variable of type default `INTEGER`, that holds the number of iterations in the current minimization at the considered level.

`tr_radius`	is a scalar variable of type default `REAL`, that holds the current level trust-region radius.
`eps_g`	is a scalar variable of type default `REAL`, that holds the current level required accuracy on the criticality measure.
`level_up`	is a pointer to the `RMTR_problem_type` structure of the finer level (that is the structure of level $i + 1$).
`level_down`	is a pointer to the `RMTR_problem_type` structure of the coarser level(that is the structure of level $i - 1$).

All this data is allocated and initialized by the subroutine `RMTR_initialize`. Note that no memory is allocated when it is not needed. Note also that other quantities that are used inside the algorithm are also stored in this structure.

D.1.3.2 The derived data type for holding control parameters

The derived data type `RMTR_control_type` is used to hold controlling data. Default values may be obtained by calling `RMTR_initialize` (whose header is given in Section D.1.4), but individual components may also be changed in this routine by reading a specification file (see Section D.1.7). The different parameters controlling the method are:

`error-printout-device` is a scalar variable of type default `INTEGER`, that holds the unit number associated with the device used for error output. The default is `error-printout-device = 6`.

`printout-device` is a scalar variable of type default `INTEGER`, that holds the unit number associated with the device used for output. The default is `printout-device = 6`.

`print-level` is a symbolic variable, that holds the level of printout requested by the user. The variable may take the values, `GALAHAD_SILENT`, `GALAHAD_SUMMARY`, `GALAHAD_TRACE`, `GALAHAD_ACTION`, `GALAHAD_DETAILS`, `GALAHAD_DEBUG` and `GALAHAD_CRAZY`. These values are described in details in Section D.1.8. The default is `print-level = GALAHAD_TRACE`.

`start-printing-at-iteration` is a scalar variable of type default `INTEGER`, that holds the index of the first RMTR iteration at which printing must occur. The default is `start-printing-at-iteration = 0` (print from initialization on).

`stop-printing-at-iteration` is a scalar variable of type default `INTEGER`, that holds the index of the last RMTR iteration at which printing must occur. If negative, printing does not stop once started. The default is `stop-printing-at-iteration = -1` (always print once started).

display-equivalent-evaluations is a scalar variable of type default LOGICAL that has the value .TRUE. iff the program needs to print a summary of the total equivalent amount of finest level work at the end of the algorithm. Note that this summary is printed only if the print-level parameter is larger than GALAHAD_SUMMARY. The default is .TRUE..

display-options is a scalar variable of type default LOGICAL that has the value .TRUE. iff the program needs to print the options used by the method at the start of the RMTR_initialize subroutine. Note that the options are printed only if the print-level parameter is larger than GALAHAD_SUMMARY. The default is .TRUE..

save-solution is a scalar variable of type default LOGICAL that has the value .TRUE. iff the program needs to store the solution of the problem in a file which has to be specified by the user in the problem specification file with the parameter solution-file (see Section D.1.5.2). The solution is saved at the start of the RMTR_terminate subroutine. The default is .TRUE..

criticality-threshold is a scalar variable of type default REAL, that specifies an accuracy threshold such that the RMTR iteration is successfully terminated if the criticality measure on the finest level is under the threshold. The default is criticality-threshold = 10^{-6}.

truncated-conjugate-gradient-accuracy is a scalar variable of type default REAL, that specifies an accuracy threshold such that the PTCG minimization of the Taylor model is successfully terminated if the model gradient is under the threshold. The default is truncated-conjugate-gradient-accuracy = 10^{-1}.

maximum-number-of-iterations is a scalar variable of type default INTEGER, that holds the maximum number of cycles during a call to RMTR_solve. The default is maximum-number-of-iterations = 1000.

maximum-number-of-tcg-iterations is a scalar variable of type default INTEGER, that holds the maximum number of truncated-conjugate-gradient iterations during a minimization of the Taylor model. The default is maximum-number-of-tcg-iterations = 5.

maximum-solving-time is a scalar variable of type default REAL, that specifies the maximum amount of time (in seconds) allowed during a call to RMTR_solve. If negative, no upper limit is imposed. The default is maximum-solving-time = 3600.

minimum-rho-for-successful-iteration is a scalar variable of type default REAL, that holds the minimum ratio of achieved vs. predicted reduction for declaring a RMTR iteration successful. The default is minimum-rho-for-successful-iteration = 0.01.

minimum-rho-for-very-successful-iteration is a scalar variable of type default REAL, that holds the minimum ratio of achieved vs. predicted reduction for declaring a RMTR iteration very successful. The default is `minimum-rho-for-very-successful-iteration = 0.9`.

radius-reduction-factor is a scalar variable of type default REAL, that holds the radius reduction factor in the case of an unsuccessful iteration. The default is `radius-reduction-factor = 0.25`.

radius-increase-factor is a scalar variable of type default REAL, that holds the radius increase factor in the case of a successful iteration. The default is `radius-increase-factor = 2.0`.

maximum-radius-increase-factor is a scalar variable of type default REAL, that holds the radius increase factor in the case of a very successful iteration. The default is `maximum-radius-increase-factor = 3.0`.

maximum-radius is a scalar variable of type default REAL, that holds the maximum trust-region radius. If negative, no upper limit is imposed. The default is `maximum-radius = -1.0`.

initial-radius is a scalar variable of type default REAL, that holds the initial trust-region radius. The default is `initial-radius = 1.0`.

forced-hessian-estimation-frequency is a scalar variable of type default INTEGER, that holds the maximum number of iterations allowed without recomputing the Hessian. Indeed, since computing the objective Hessian is commonly one of the heaviest task of optimization algorithms, RMTR features a strategy that avoids recomputing the Hessian at each iteration (see (Gratton et al. 2009)). If zero, no forced evaluation is made except by the automatic criterion. The default is `forced-hessian-estimation-frequency = 0`.

forced-hessian-evaluation-factor is a scalar variable of type default REAL, that holds the minimum ratio of achieved vs. predicted reduction under which Hessian evaluation is forced. The default is `forced-hessian-evaluation-factor = 0.5`.

euclidean-gradient-accuracy-for-hessian-evaluation is a scalar variable of type default REAL, that holds the minimum relative accuracy of the predicted gradient (in Euclidean norm) under which Hessian reevaluation is forced. The default is `euclidean-gradient-accuracy-for-hessian-evaluation = 0.15`.

infinite-gradient-accuracy-for-hessian-evaluation is a scalar variable of type default REAL, that holds the minimum relative accuracy of the predicted gradient (in infinity norm) under which Hessian reevaluation is forced. The default is `infinite-gradient-accuracy-for-hessian-evaluation = 10000`.

initialization-technique is a symbolic variable, that holds the multilevel strategy used to solve the problem (see (Gratton et al. 2009)). The parameter may take the values GALAHAD_AF (All on Finest algorithm), GALAHAD_MR (Mesh Refinement algorithm), GALAHAD_FM (Full Multilevel algorithm), GALAHAD_MF (Multilevel on Finest algorithm) and GALAHAD_FMF (Full Multilevel on Finest algorithm). If algorithm GALAHAD_AF, algorithm GALAHAD_MF or algorithm GALAHAD_FMF is chosen, the user only needs to provide information (objective function, gradient, ...) about the finest level. Otherwise, the user has to provide information for all levels. The default is `initialization-technique = GALAHAD_FM`.

cycling-style is a symbolic variable, that holds the type of recursion done by the algorithm. The parameter may take the values GALAHAD_Wcycles (W-cycles), GALAHAD_Vcycles (Vcycles) or, GALAHAD_freecycles (recursion is finished only when accuracy is reached). The default value is `cycling-style = GALAHAD_Vcycles`.

coarse-model-choice-parameter is a scalar variable of type default REAL, that holds the minimum ratio of coarser linear decrease vs current level linear decrease over which minimization of the coarser local model is preferred.The default value is `coarse-model-choice-parameter= 0.25`.

linesearch is a scalar variable of type default INTEGER, that holds the maximum number of additional function evaluations allowed for a linesearch procedure. The default is `linesearch= 2`.

model-backtracking is a scalar variable of type default LOGICAL, that is .TRUE. iff backtracking of the model along the current step is allowed after an unsuccessful iteration. The default is `model-backtracking = .TRUE.`.

quadratic-model is a symbolic variable, that holds the type of coarser local model used by the algorithm. The different possible values are GALAHAD_FIRST_ORDER for a first-order coherent coarse local model, GALAHAD_SECOND_ORDER for a second-order coherent coarse local model and GALAHAD_GALERKIN which is a restricted version of the current level quadratic Taylor model. The default is `quadratic-model = GALAHAD_GALERKIN`.

number-of-smoothing-cycles is a scalar variable of type default INTEGER, that holds the number of smoothing cycles allowed at each minimization of the Taylor model by a Sequential Coordinate Minimization method (see (Gratton et al. 2009)). The default is `number-of-smoothing-cycles = 7`.

smooth-frequency is a scalar variable of type default INTEGER, that holds the frequency of smoothing along the recursions. The possible values are GALAHAD_NEVER_SMOOTH if no smoothing is done before or after a recursion, GALAHAD_SMOOTH_UP if SCM smoothing is imposed after a recursion, GALAHAD_SMOOTH_DOWN if SCM smoothing is imposed before a recursion and GALAHAD_ALWAYS_SMOOTH if SCM smooth-

ing is imposed before and after a recursion,.The default is `smooth-frequency = GALAHAD_ALWAYS_SMOOTH`.

`checkpointing-frequency` is a scalar variable of type default `INTEGER`, that holds the frequency (expressed in number of finest iterations) at which the current values of the problem's variables and the trust-region radius are saved on a checkpointing file for a possible package restart. It must be non-negative. The default is `checkpointing-frequency = 0` (no checkpointing).

`checkpointing-file` is a scalar variable of type default `CHARACTER` of length 30, that holds the name of the file use for storing checkpointing information on disk. The default is `checkpointing-file = RMTR.sav`.

`checkpointing-device` is a scalar variable of type default `INTEGER`, that holds the number of the device that must be used for input/output of checkpointing operations. The default is `checkpointing-device = 55`.

`restart-from-checkpoint` is a scalar variable of type default `LOGICAL`, whose value is `.TRUE.` iff the initial point must be read from the checkpointing file, overriding the input value of the starting point. The default is `restart-from-checkpoint = .FALSE.`.

D.1.3.3 The derived data type for holding informational parameters

The derived data type `RMTR_inform_type` is used to hold parameters that give information about the progress and needs of the algorithm. The components of `RMTR_inform_type` are:

`status` is a scalar variable of type default `INTEGER`, that gives the exit status of the algorithm. See Section D.1.6 for details.

`message` is a character array of 3 lines of 80 characters each, containing a description of the exit condition on exit, typically including more information than contained in status. It is printed out on device `error-printout-device` at the end of execution unless print level is `GALAHAD_SILENT`.

`nbr_taylor` is a rank one array of type default `INTEGER`, and which length is the number of levels where each component is the number of Taylor minimizations at this level, that is the number of minimizations of the Taylor model by a PTCG procedure.

`nbr_taylor_iterations` is a rank one array of type default `INTEGER`, and which length is the number of levels where each component is the total number of Taylor iterations at this level, that is the total number of PTCG iterations at this level.

`nbr_smoothing` is a rank one array of type default `INTEGER`, and which length is the number of levels where each component is the number of SCM minimizations at this level, that is the number of minimizations of the Taylor model by the SCM procedure.

nbr_cycles is a rank one array of type default INTEGER, and which length is the number of levels where each component is the total number of SCM cycles at this level, that is the total number of SCM iterations at this level.

nbr_backtrackings is a rank one array of type default INTEGER, and which length is the number of levels where each component is the number of backtracking iterations at this level.

nbr_f_evaluations is a rank one array of type default INTEGER, and which length is the number of levels where each component is the number of function evaluations at this level.

nbr_g_evaluations is a rank one array of type default INTEGER, and which length is the number of levels where each component is the number of gradient evaluations at this level.

nbr_H_evaluations is a rank one array of type default INTEGER, and which length is the number of levels where each component is the number of Hessian evaluations at this level.

nbr_H_updates is a rank one array of type default INTEGER, and which length is the number of levels where each component is the number of Hessian LTS updates at this level, that is, the number of times where the Hessian values are recomputed (by evaluating gradients, the structure is never recomputed).

nbr_H_reductions is a rank one array of type default INTEGER, and which length is the number of levels where each component is the number of Hessian reductions at this level, that is the component i of this array contains the number of Hessian reductions from level i to level $i - 1$.

nbr_prolongations is a rank one array of type default INTEGER, and which length is the number of levels where each component is the number of prolongations from this level to the finer one, that is the component i of this array contains the number of vector prolongations from level i to level $i + 1$.

nbr_restrictions is a rank one array of type default INTEGER, and which length is the number of levels where each component is the number of restrictions from this level to the coarser one, that is the component i of this array contains the number of vector restrictions from level i to level $i - 1$.

nbr_projections is a rank one array of type default INTEGER, and which length is the number of levels where each component is the number of projections on the constraints at this level.

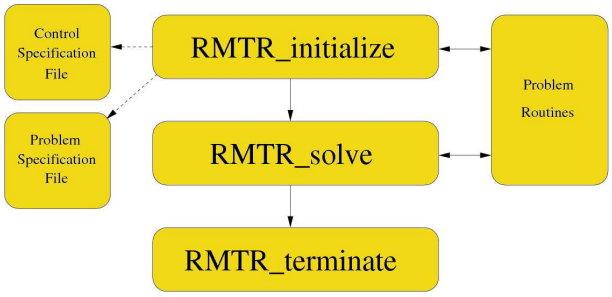

Figure D.1: Calling sequence of the RMTR package

D.1.4 Argument lists and calling sequences

Access to the package requires a USE statement such as

```
USE RMTR
```

The package uses two specification files. The first one is a control file whose purpose is to modify the default values of the algorithmic parameters of the method. The second file defines important characteristics of the problem, such as its dimension, . . . (the complete list of the problem parameters is described in Section D.1.5.2). The description and syntax of these specification files is available in Section D.1.7.

The actual call to the package consists of three successive subroutine calls, as illustrated in Figure D.1.

1. The subroutine `RMTR_initialize` is first called by the statement

```
CALL RMTR_initialize( control,inform,problem,algospecs,
  probspecs [, MY_GRAD=GRAD] [, MY_STRUCT=STRUCT]
  [,MY_SPARSITY=SPARSITY] [, MY_LOWER_BOUND=LOWER_BOUND]
  [,MY_UPPER_BOUND=UPPER_BOUND]
  [,MY_PROLONGATION=PROLONGATION]
  [,MY_RESTRICTION=RESTRICTION] [, MY_SIZE=SIZE] )
```

where `control` is a scalar argument of type `RMTR_control_type`,`inform` is a scalar argument of type `RMTR_inform_type`, `problem` is a pointer of type `RMTR_problem_type` and `algospecs` and `probspecs` are two strings containing the name of the control and problem specification files respectively. The other arguments are optional and are used to provide the algorithm routines describing the problem (`GRAD` is the user subroutine to compute the objective gradient, `STRUCT` is the user subroutine to compute the Hessian structure, . . .). They are all described more in details in Section D.1.5. On exit, `control` contains default values for all the components that may be changed by the values given by the user in the control specification file and all components of the

multilevel structure are allocated according to the values provided by the user in the problem specification file. A successful call to the routine `RMTR_initialize` is indicated when the component `status` of the `inform` argument has the value 0.

2. The subroutine `RMTR_solve` is called by

```
CALL RMTR_solve( control, inform, problem , MY_FUN=FUN ,
  MY_GRAD=GRAD [, MY_HESS=HESS] )
```

where `control` is a scalar argument of type `RMTR_control_type`,`inform` is a scalar argument of type `RMTR_inform_type`, `problem` is a pointer of type `RMTR_problem_type`. The other arguments are routines to describe the problem. `FUN` is the user subroutine to compute the objective function `GRAD` is the user subroutine to compute the objective gradient and `HESS` is the user subroutine to compute the objective Hessian and is optional. The headers of these routines are described in Section D.1.5. This routine is called to solve the problem by applying the RMTR algorithm. A call to this routine must be preceded by a call to the `RMTR_initialize` routine. On exit, the solution is in the `problem%x` component. A successful call to the routine `RMTR_solve` is indicated when the component `status` of the `inform` argument has the value 0. For other return values of the `status` component, see Section D.1.6. The other `inform` components contain the iterations history of the algorithm as explained in Section D.1.3.3.

The starting point of the algorithm is also determined at the start of `RMTR_solve`, either from the user-specified values which are read in a file (the file name is specified in the problem specification file by the parameter `starting-point-file` and each line of this file has to contain the corresponding component of the starting point), or, if no such file exist, by the following simple component-wise procedure.

(a) If both lower and upper bounds are given, the starting point is computed as the mid-point between these bounds.

(b) If only a lower or upper bound is given, the starting point is computed at a distance of 1 (in infinity norm) of the specified constraint.

(c) If no bound is given, the starting point is set to 1.

(d) A small random perturbation is then applied. The perturbation is uniform and of amplitude 10^{-2}.

Note that the starting point is defined at the finest level, irrespective of the actual multilevel technique used to solve the problem (if a mesh refinement or a full multilevel technique is chosen by the user, the starting point is first restricted to the coarsest level).

3. The subroutine `RMTR_terminate` is called by

```
CALL RMTR_terminate( control, inform, problem )
```

where `control` is a scalar argument of type `RMTR_control_type`,`inform` is a scalar argument of type `RMTR_inform_type`, `problem` is a pointer of type `RMTR_problem_type`. This routine automatically deallocates the RMTR-specific arrays. A call to this routine must be preceded by a call to the `RMTR_initialize` routine. The solution is also written in a file if required by the user by the control parameter `save-solution`. In this case, each line of the file contains the corresponding component of the solution. A summary print of the iterations is also printed if the control parameter `display-summary` is set.

D.1.5 Information needed by the algorithm

Many multilevel problems are discretizations of an underlying infinite-dimensional problem on a grid. RMTR thus features the necessary routines for this class of problems. The sections below describe these routines and how an user has to create his routines.

D.1.5.1 Important note

Note that if the user has chosen to use the Mesh Refinement algorithm or the Full Multilevel algorithm, the problem has to be described for different levels. This means that the user has to write the routines `FUN`, `GRAD`, `HESS`, `STRUCT`, `SPARSITY`, `LOWER_BOUND` and `UPPER_BOUND` in such a way that they may be evaluated from different levels. The knowledge of this hierarchy information is important to derive the mesh refinement strategy used to obtain better starting points that leads to better numerical results. However, if this knowledge is impossible to obtain the user has to use one of the other three algorithms provided in the RMTR package. Note that the level for which the routine is computed is not an input of the routines since it is possible to derive it with for example the size of the input vectors.

D.1.5.2 The problem specification

In the same spirit of the parameters of the control structure presented above, the user may provide information about his problem by a problem specification file. The different possible parameters are:

`problem-dimension` is a scalar variable of type default `INTEGER`, that holds the problem dimension. This variable does not correspond to the variable dimension but corresponds to the dimension of the space where the problem is considered. This parameter is only needed for geometric problems. The default is `problem-dimension = 2`.

`level-min` is a scalar variable of type default `INTEGER`, that holds the number of the coarsest level. The default is `level-min=1`.

`level-max` is a scalar variable of type default `INTEGER`, that holds the number of the finest level. The default is `level-max=4`.

interpolation-type is a symbolic variable, that holds the type of transfer operators used. The different possible values are GALAHAD_USER if the transfer operators are provided by the user, GALAHAD_LINEAR if linear interpolation operators are always used, GALAHAD_LINEAR_CUBIC if linear interpolation operators are always used except for prolongating the solution of a level to a starting point of the upper level in the mesh-refinement process where a cubic interpolation operator is used and GALAHAD_CUBIC if cubic interpolation operators are always used. Note that the last three choices are only valid for geometric problems. The default is `interpolation-type = GALAHAD_LINEAR_CUBIC`.

matrix-storage is a symbolic variable, that holds the type of storage used to store the objective Hessian. Note that whatever the type of storage used by the user to compute the objective Hessian, it is transformed after the Hessian computation in a compressed sparse row storage. The possible values are GALAHAD_COORDINATE for a sparse co-ordinate storage and GALAHAD_SPARSE_BY_ROWS for a compressed sparse row storage. The default is `matrix-storage = GALAHAD_COORDINATE`.

number-of-field-variables is a scalar variable of type default INTEGER, that holds the number of field variables of the problem, that is the number of fields to compute for the problem. This parameter is only needed for geometric problems. The default is `number-of-field-variables = 1`.

number-of-coarsest-level-discretization-points is a scalar variable of type default INTEGER, that holds the number of coarsest level discretization free points (geometric problems are supposed to have fixed boundaries). This parameter is only needed for geometric problems. The default is `number-of-coarsest-level-discretization-points = 9`.

half-hessian is a scalar variable of type default LOGICAL, that has the value `.TRUE.` iff only the half hessian is computed and is stored. Note that the algorithm is in this case less efficient but requires less memory. The default is `half-hessian = .FALSE.`.

upper-bound is a scalar variable of type default LOGICAL, that has the value `.TRUE.` iff the problem admits an upper bound. The default is `upper-bound = .FALSE.`.

lower-bound is a scalar variable of type default LOGICAL, that has the value `.TRUE.` iff the problem admits a lower bound. The default is `lower-bound = .FALSE.`.

quadratic-problem is a scalar variable of type default LOGICAL, that has the value `.TRUE.` iff the problem is a quadratic problem and thus, admits a constant Hessian that has not to be recomputed. The default is `quadratic-problem = .FALSE.`.

starting-point-file is a scalar variable of type default CHARACTER of length 30, that holds the name of the file used to store the starting point. The default is `starting-point-file = RMTR_startingpoint.dat`.

`solution-file` is a scalar variable of type default `CHARACTER` of length 30, that holds the name of the file used to store the solution. The default is `solution-file = RMTR_solution.dat`.

`approximate-hessian` is a symbolic variable, that holds the type of Hessian used by the problem. The different possible values are `GALAHAD_EXACT_HESSIAN` if the exact Hessian is used, `GALAHAD_LTS_STRUCT` if a lower-triangular substitution (LTS) Hessian is used for which the user provide the partial separability structure (see Section D.1.5.5 for a description of the partial separability structure), `GALAHAD_LTS_SPARSITY` if a lower-triangular substitution (LTS) Hessian is used for which the user provide the sparsity pattern of the Hessian (the vectors `H_row` and `H_col` stored in a compressed sparse row format) and `GALAHAD_LTS_PREDEFINED_PATTERN` if a lower-triangular substitution (LTS) Hessian is used for which the sparsity pattern is computed by an already implemented pattern specified by the option `predefined-sparsity-pattern` (see Section D.1.5.5 for more details). The default value is `approximate-hessian = GALAHAD_EXACT_HESSIAN`.

`predefined-sparsity-pattern` is a scalar variable of type default `INTEGER`, that holds the type sparsity pattern already implemented used to approximate the objective Hessian. The different patterns are

1. 5-diagonal two-dimensional Hessian $(-\sqrt{n}, -1, 0, 1, \sqrt{n})$
2. 7-diagonal two-dimensional Hessian $(-\sqrt{n}-1, -\sqrt{n}, -1, 0, 1, \sqrt{n}, \sqrt{n}+1)$
3. 7-diagonal two-dimensional Hessian $(-\sqrt{n}+1, -\sqrt{n}, -1, 0, 1, \sqrt{n}, \sqrt{n}-1)$
4. 13-diagonal two-dimensional Hessian$(-2\sqrt{n}, -\sqrt{n}-1, -\sqrt{n}, -\sqrt{n}+1, -2, -1, 0, 1, 2, \sqrt{n}-1, \sqrt{n}, \sqrt{n}+1, 2\sqrt{n})$
5. 7-diagonal three-dimensional Hessian $(-n^{\frac{2}{3}}, -n^{\frac{1}{3}}, -1, 0, 1, n^{\frac{1}{3}}, n^{\frac{2}{3}})$

where the number between the parenthesis represent the indexes of the subdiagonals and superdiagonals where the only nonzero elements are present. By convention, 0 is the diagonal, positive numbers correspond to the index of the band above the diagonal and negative numbers to the index of the band below the diagonal. The patterns already implemented correspond to discretizations of frequently-used multilevel problems (for example, the first pattern is the pattern of a two-dimensional Laplacian).

D.1.5.3 The objective function

Obviously the algorithm needs to compute the value of the objective function at a given iterate. This routine has to be given to the algorithm as described above. The header of the subroutine has to be the following:

```
SUBROUTINE FUN(x,f)
```

where,

x is an `INTENT(IN)` array of type default `REAL` that holds the point where the objective function is computed.

f is an `INTENT(OUT)` scalar of type default `REAL` that holds the computed objective value.

D.1.5.4 The objective gradient

The algorithm also needs to compute the value of the objective gradient at a given iterate. This has to be done by a routine provided by the user which header has to be the following:

```
SUBROUTINE GRAD(x, g)
```

where,

x is an `INTENT(IN)` array of type default `REAL` that holds the point where the objective gradient is computed.

g is an `INTENT(INOUT)` array of type default `REAL` that holds the objective gradient computed.

D.1.5.5 The objective Hessian

Computing the objective Hessian is commonly one of the heaviest task of optimization algorithms. RMTR thus features a flexible strategy that allows the user to approximate it by different ways using a lower-triangular substitution (LTS) technique.

If it is easy to compute the objective hessian, the user may provide a routine where the objective Hessian is computed and stored in a sparse way. The header of the routine in this case has to be:

```
SUBROUTINE HESS(x, Hval, Hrow, Hcol, Hnz)
```

x is an `INTENT(IN)` array of type default `REAL` that holds the point where the objective Hessian is computed.

`Hval, Hrow, Hcol` are pointers of type default `REAL`, default `INTEGER` and default `INTEGER` , respectively. They hold the objective Hessian stored using a sparse storage type. Note that these are pointers since it may be necessary to deallocate them and then to allocate them with an appropriate size.

Hnz is an `INTENT(OUT)` scalar of type default `INTEGER` that holds the number of nonzero elements in the objective Hessian.

On the other hand, if it is not easy to compute the Hessian, the algorithm may use a lower-triangular substitution (LTS) Hessian. The user must in this case provide the sparsity structure of the Hessian. This may be done in several ways:

1. A lot of multilevel problems have a partially separable structure. This means that the objective function f is the sum of element functions f_k $(k = 1, \ldots, k_{\max})$ each depending of only few variables. The RMTR package features the use of such functions and the user has only to provide the partial separability decomposition of the Hessian by the routine `STRUCT`

   ```
   SUBROUTINE STRUCT(nbvar, nbrelem, xelvar, elvar)
   ```

   where,

   `nbvar` is an `INTENT(IN)` scalar of type default `INTEGER` that holds the size of the problem variables at the current level.

   `nbrelem` is an `INTENT(OUT)` scalar of type default `INTEGER` that holds the number of element functions in the partial separability definition of the objective function.

   `xelvar,elvar` are pointers of type default `INTEGER` that hold the partial separability structure of the objective Hessian in a framework related to the compressed sparse row storage for sparse matrices. The l-th entry of the vector `xelvar` contains the index of the first component in the vector `elvar` where an element of the l-th function is stored and the vector `elvar` contains the list of the variables indexes used by each separable function. The last element of `xelvar` is the size of `elvar` + 1.

2. The user may provide the sparsity structure of the Hessian by the routine `SPARSITY`

   ```
   SUBROUTINE SPARSITY(nbvar, Hnz, srow, scol)
   ```

   where,

   `nbvar` is an `INTENT(IN)` scalar of type default `INTEGER` that holds the size of the problem variables at the current level.

   `Hnz` is an `INTENT(OUT)` scalar of type default `INTEGER` that holds the number of nonzero elements in the objective Hessian.

   `srow,scol` are pointers of type default `INTEGER` that hold the sparsity structure of the objective Hessian. That is, it corresponds to the row and column vectors of a compressed sparse row storage of the Hessian.

3. Or finally, the user could finally use one of the specified sparsity pattern already implemented by using the `predefined-sparsity-pattern` option (see Section D.1.5.2).

Note that the user need only to provide either one of the three routines (`HESS`, `STRUCT` or `SPARSITY`) or specify the already implemented sparsity pattern to use. This has to be made by specifying the `approximate-hessian` parameter in the problem specification file.

D.1.5.6 The bound constraints

The user may also specify the bound constraints used by the problem by providing the routines `LOWER_BOUND` and `UPPER_BOUND`:

```
SUBROUTINE LOWER_BOUND(lb)

SUBROUTINE UPPER_BOUND(ub)
```

`lb` is an `INTENT(INOUT)` array of type default `REAL` that holds the lower bound-constraints at the finest level.

`ub` is an `INTENT(INOUT)` array of type default `REAL` that holds the upper bound-constraints at the finest level.

Note that it is not necessary to construct these subroutines if the problem admits no bound constraints. The constraints are considered only if the parameter `upper-bound` and/or `lower-bound` are set to `.TRUE.` in the problem specification file.

D.1.5.7 The transfer operators

The user may use one of the already implemented transfer operators if the problem is a "geometric" problem with fixed boundary. Otherwise, the user has to specify his own transfer operators by the routines `PROLONGATION` and `RESTRICTION`. Let consider the i-th level for which the number of variables is `nbvar`. The headers of these routines are

```
SUBROUTINE PROLONGATION( P_val, P_row, P_col, p, q, nbvar )

SUBROUTINE RESTRICTION( R_val, R_row, R_col, p, q, nbvar,sig-
ma )
```

where,

`P_val,P_row,P_col` are pointers of type default `REAL`, default `INTEGER` and default `INTEGER` , respectively. They hold the prolongation operator from level i to the finer level $i + 1$ stored using a compressed sparse row storage.

`R_val,R_row,R_col` are pointers of type default `REAL`, default `INTEGER` and default `INTEGER` , respectively. They hold the restriction operator from level i to the coarser level $i - 1$ stored using a compressed sparse row storage.

`p,q` are `INTENT(OUT)` scalars of type default `INTEGER` that hold the number of lines and the number of columns , respectively, of the operator.

`nbvar` is an `INTENT(IN)` scalar of type default `INTEGER` that holds the size of the variables of level i.

sigma is an INTENT(OUT) scalar of type default REAL that holds 1-norm of the restriction operator from level i to level $i-1$.

The operators are then only consider if the parameter interpolation-type is set to GALAHAD_USER in the problem specification file.

D.1.5.8 The levels size

And finally, the user has to provide, if the problem is not a "geometric" problem, the size of the variables at each level. This has to be done by the routine called SIZE which header is the following

```
SUBROUTINE SIZE(size)
```

where,

size is an INTENT(INOUT) array of type default INTEGER that holds the size of the variables for each level.

This routine is only considered in the case of user-specified transfer operators, which indicates that the problem is not a geometric one.

D.1.6 Warning and error messages

A negative value of inform%status on exit from RMTR_initialize, RMTR_solve or RMTR_terminate indicates that an error has occurred. No further calls should be made to these routines until the error has been corrected. Possible values are:

-1 The memory allocation failed.

-2 A file cannot be opened.

-3 Impossible to write into a file.

-4 Impossible to read a file.

-6 Wrong input (negative level for example).

-7 A vector has a wrong size.

-9 An attempt to restrict a vector from the coarsest level was made.

-10 An attempt to prolongate a vector from the finest level was made.

-21 An IO error occurred while saving checkpointing information on the relevant disk file.

-23 An input is not associated.

-29 inform%status is not correct.

-30 The maximum number of iterations has been reached and computation terminated.

-31 Further progress of the algorithms appears to be impossible, although successful termination is not recognized.

Whatever, the error status, more information could be obtained by printing `inform%message`. Note that when the value of a parameter in a specification file is not correct, a warning message is printed although the algorithm still continues with the default value of the parameter replacing the wrong value.

D.1.7 The control and problem specification files

In this section, an alternative means of setting control parameters is described, that is components of the variable control of type `RMTR_control_type` (see Section D.1.3.2), by reading an appropriate data specification file. This facility is useful as it allows a user to change RMTR control parameters without editing and recompiling programs that call RMTR.

A specification file, or specfile, is a data file containing a number of 'specification commands'. Each command occurs on a separate line, and comprises a 'keyword', which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) 'value', which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by RMTR must start with a `BEGIN RMTR` command and ends with an `END` command. The syntax of the specfile is thus defined as follows:

```
( ..  lines ignored ..  )
BEGIN RMTR
 keyword  value
 .......  .....
 keyword  value
END
( ..  lines ignored ..  )
```

where keyword and value are two strings separated by (at least) one blank. The `BEGIN RMTR` and `END` delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN RMTR SPECIFICATION
```

and

```
END RMTR SPECIFICATION
```

are acceptable. Furthermore, between the `BEGIN RMTR` and `END` delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy manner to 'comment off' some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of five different types, namely integer, logical, real, string or symbol. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are `ON`, `TRUE`, `.TRUE.`, `T`, `YES`, `Y`, or `OFF`, `NO`, `N`, `FALSE`, `.FALSE.` and `F`. Empty values are also allowed for logical control parameters, and are interpreted as `TRUE`. String are specified as a sequence of characters. A symbolic value is a special string obtained from one of the predefined symbols of the `SYMBOLS` module by deleting the leading '`GALAHAD_`' characters in its name. Thus, the specification command

```
print-level SILENT
```

implies that the value `GALAHAD_SILENT` is assigned to `control%print-level`. This technique is intended to help expressing an (integer) control parameter for an algorithm in a 'language' that is close to natural. A default specification file is in the `src` directory. The complete list of parameters is enclosed in the tables D.2 (control parameters) and D.3 (problem parameters) with their types and default values.

Name of the parameter	Value type/Symbolic value	Default value
error-printout-device	INTEGER	6
printout-device	INTEGER	6
print-level	SILENT,	
	SUMMARY,	
	TRACE,	
	ACTION,	TRACE
	DETAILS,	
	DEBUG,	
	CRAZY	
start-printing-at-iteration	INTEGER	0
stop-printing-at-iteration	INTEGER	-1
display-equivalent-evaluations	LOGICAL	.TRUE.
display-options	LOGICAL	.TRUE.
save-solution	LOGICAL	.TRUE.
criticality-threshold	REAL	10^{-6}
truncated-conjugate-gradient-accuracy	REAL	10^{-1}
maximum-number-of-iterations	INTEGER	1000
maximum-number-of-tcg-iterations	INTEGER	5
maximum-solving-time	REAL	3600.0
minimum-rho-for-successful-iteration	REAL	0.01
minimum-rho-for-very-successful-iteration	REAL	0.9
radius-reduction-factor	REAL	0.25
radius-increase-factor	REAL	2.0
maximum-radius-increase-factor	REAL	3.0
maximum-radius	REAL	-1.0
initial-radius	REAL	1.0
forced-hessian-estimation-frequency	INTEGER	0
forced-hessian-evaluation-factor	REAL	0.5
euclidean-gradient-accuracy-for-hessian-evaluation	REAL	0.15
infinite-gradient-accuracy-for-hessian-evaluation	REAL	10000.0
initialization-technique	AF,	

Name of the parameter	Value type/Symbolic value	Default value
	MR,	
	FM,	FM
	MF,	
	FMF	
cycling-style	Vcycles,	
	Wcycles,	Vcycles
	freecycles	
coarse-model-choice-parameter	REAL	0.25
linesearch	INTEGER	2
model-backtracking	LOGICAL	.TRUE.
quadratic-model	FIRST_ORDER,	
	SECOND_ORDER,	GALERKIN
	GALERKIN	
number-of-smoothing-cycles	INTEGER	7
smooth-frequency	NEVER_SMOOTH,	
	SMOOTH_DOWN,	
	SMOOTH_UP,	ALWAYS_SMOOTH
	ALWAYS_SMOOTH	
checkpointing-frequency	INTEGER	0
checkpointing-file	CHARACTER	RMTR.sav
checkpointing-device	INTEGER	55
restart-from-checkpoint	LOGICAL	.FALSE.

Table D.2: Control parameters with their types and default values.

Name of the parameter	Value type/Symbolic value	Default value
problem-dimension	INTEGER	2
level-min	INTEGER	1
level-max	INTEGER	4
interpolation-type	USER,	
	LINEAR,	
	LINEAR_CUBIC,	LINEAR_CUBIC
	CUBIC	
matrix-storage	COORDINATE,	
	SPARSE_BY_ROWS	COORDINATE
number-of-field-variables	INTEGER	1
number-of-coarsest-level-discretization-points	INTEGER	9
half-hessian	LOGICAL	.FALSE.
upper-bound	LOGICAL	.FALSE.
lower-bound	LOGICAL	.FALSE.
quadratic-problem	LOGICAL	.FALSE.
starting-point-file	CHARACTER	RMTR_startingpoint.dat
solution-file	CHARACTER	RMTR_solution.dat
approximate-hessian	EXACT_HESSIAN,	
	LTS_STRUCT,	
	LTS_SPARSITY,	EXACT_HESSIAN
	LTS_PREDEFINED_PATTERN	
predefined-sparsity-pattern	INTEGER	0

Table D.3: Problem parameters with their types and default values.

D.1.8 Information printed

The meaning of the various `control%print_level` values is defined as follows:

`GALAHAD_SILENT:` nothing is printed.

`GALAHAD_SUMMARY:` only reports a summary of the iterations at the end of the algorithm.

GALAHAD_TRACE: reports a one line summary of each iteration. This summary includes the current level number, the number of variables of the current level, the current number of iterations at this level, the current values of the objective function, the criticality measure value, the step norm, the trust-region radius, the ratio of achieved to predicted reduction, the iteration type, the model decrease and the objective function decrease. The iteration type is a six character string which can be `'LOWER '` if the algorithm just starts a recursion, `'TAYLOR'` if the algorithm minimizes the Taylor model using a standard trust-region technique, `'SMOOTH'` if the algorithm minimizes the Taylor model using a smoothing technique, `'BACKTR'` if the algorithm makes a backtracking iteration or `'UPPER '` if the algorithm just finishes a recursion.

GALAHAD_ACTION: reports the mains steps of each iteration.

GALAHAD_DETAILS, GALAHAD_DEBUG and GALAHAD_CRAZY report more and more information.

Note that, after the summary of the iterations, the algorithm reports a summary of the equivalent iterations, that is the total amount of work expressed as work in the finest level (see (Gratton et al. 2009)). This measure is a better measure of the total amount of work done at all levels. The equivalent quantities reported are:

f evaluations corresponds to the equivalent number of objective function evaluations.

g evaluations corresponds to the equivalent number of objective gradient evaluations.

H evaluations corresponds to the equivalent number of objective Hessian evaluations

smoothing cycles corresponds to the equivalent number of SCM smoothing cycles.

Taylor iterations corresponds to the equivalent number of PTCG iterations.

H updates corresponds to the equivalent number of Hessian updates (in the case of approximate Hessians), that is, the number of times where the Hessian values are recomputed (by evaluating gradients, the structure is never recomputed).

H eval+upd corresponds to the sum of the equivalent number of Hessian evaluations and updates.

The algorithm finally reports the total amount of solving time and the total amount of time to solve the problem (including the time spent to construct the multilevel structure, the transfer operators, ...).

D.2 Example of use

Consider the minimum surface problem

$$\min_{v \in \mathcal{K}} \int_{S_2} \sqrt{1 + \|\nabla_x v\|_2^2},$$

where $\mathcal{K} = \{v \in H^1(S_2) \mid v(x) = v_0(x) \text{ on } \partial S_2\}$. This convex problem is discretized using a finite-element basis defined using a uniform triangulation of S_2, with the same grid spacing, h, along the two coordinate directions. The basis functions are the classical P1 functions which are linear on each triangle and take the value 0 or 1 at each vertex. The boundary condition $v_0(x)$ is chosen as

$$v_0(x) = \begin{cases} f(x_1), & x_2 = 0, \quad 0 \leq x_1 \leq 1, \\ 0, & x_1 = 0, \quad 0 \leq x_2 \leq 1, \\ f(x_1), & x_2 = 1, \quad 0 \leq x_1 \leq 1, \\ 0, & x_1 = 1, \quad 0 \leq x_2 \leq 1, \end{cases}$$

where $f(x_1) = x_1(1 - x_1)$. The problem has the following lower bound constraint:

$$v(x) \geq \sqrt{2} \qquad \text{whenever} \quad \frac{4}{9} \leq x_1, x_2 \leq \frac{5}{9},$$

thereby creating an obstacle problem where the surface is constrained in the middle of the domain.

The call to the RMTR package may be done by

```
PROGRAM GALAHAD_RMTR_EXAMPLE

  USE TYPES
  USE sparseroutines
  USE GALAHAD_RMTR_double

  USE GALAHAD_SYMBOLS,
&
       OK                            => GALAHAD_SUCCESS

  IMPLICIT NONE

  TYPE( RMTR_inform_type )           :: inform
  TYPE( RMTR_control_type )          :: control
  TYPE( RMTR_problem_type ), pointer :: problem

  CHARACTER( LEN = 60 ) :: specname = 'RUNRMTR.SPC'
  CHARACTER( LEN = 60 ) :: problemspecname = 'PROBLEM.SPC'
  EXTERNAL :: cost, grad, hessian,compute_lower_bound

  ALLOCATE(problem)

  CALL RMTR_initialize( control, inform, problem, specname, &
       problemspecname, MY_GRAD=grad,                       &
       MY_LOWER_BOUND=compute_lower_bound)

  IF(inform%status == OK)THEN
     CALL RMTR_solve( control, inform, problem, MY_FUN=cost, &
```

```
          MY_GRAD=grad,MY_HESS=hessian )
  END IF

  CALL RMTR_terminate( control, inform, problem )

  DEALLOCATE(problem)

END PROGRAM GALAHAD_RMTR_EXAMPLE
```

Assume that the subroutines `f0`, `f1`, `f2` and `f3` to compute the boundary conditions of the problem are implemented. The routines `cost`, `grad`, `hessian` and `compute_lower_bound` which compute the objective function, gradient and Hessian and the lower bound of the problem, respectively are given by

```
SUBROUTINE cost(x,f)
  USE TYPES, ONLY: wp

  REAL ( KIND = wp ), INTENT(IN),  DIMENSION( : ) :: x
  REAL ( KIND = wp ), INTENT(OUT) :: f

  ! Local variables
  REAL( KIND = wp ) :: a,h,p,q,r,s,c,c1,c2,lc,k
  INTEGER :: i,j,temp
  REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( :, : ) :: x_mod

  a = SIZE(x)
  a = SQRT(a)
  temp = NINT(a)
  h=1/(a+1)

  c = 0
  ALLOCATE(x_mod(temp+2,temp+2))
  x_mod = 0
  i = 0
  DO WHILE(i<temp)
     i = i+1
     x_mod(2:temp+1,i+1) = x(1+(i-1)*temp:i*temp)
  END DO

  j=0
  k=0.0
  DO WHILE(j<temp+2)
     j=j+1
     k = k + 1.0
     x_mod(1,j) = f0((k-1)/(temp+1))
     x_mod(j,1) = f1((k-1)/(temp+1))
```

```
      x_mod(temp+2,j) = f2((k-1)/(temp+1))
      x_mod(j,temp+2) = f3((k-1)/(temp+1))
   END DO

   i=0
   DO WHILE(i<temp+1)
      i=i+1
      j=0
      DO WHILE(j<temp+1)
         j=j+1
         p=x_mod(i,j)
         q=x_mod(i+1,j)
         r=x_mod(i+1,j+1)
         s=x_mod(i,j+1)
         c1=SQRT(1+ ((s-p)/h)**2    + ((s-r)/h)**2)
         c2=SQRT(1+ ((p-q)/h)**2    + ((r-q)/h)**2)
         lc=0.5*(c1+c2)*h**2
         c=c+lc
      END DO
   END DO

   f = c

   DEALLOCATE(x_mod)

END SUBROUTINE cost
```

```
SUBROUTINE grad(x, g)
   USE TYPES,ONLY : wp

   REAL ( KIND = wp ), INTENT(IN),  DIMENSION( : ) :: x
   REAL ( KIND = wp ), INTENT(OUT), DIMENSION( : ) :: g

   ! Local variables
   REAL( KIND = wp ) :: a,h,p,q,r,s,c1,c2,lc, k
   INTEGER           :: i,j,temp, pt1, pt2, pt3, pt4
   REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( :, : ) :: x_mod
   REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( : )    :: g_mod

   a = SIZE(x)
   a = SQRT(a)
   temp = NINT(a)
   h=1/(a+1)

   ALLOCATE(x_mod(temp+2,temp+2))
   x_mod = 0
```

```
i = 0
DO WHILE(i<temp)
   i = i+1
   x_mod(2:temp+1,i+1) = x(1+(i-1)*temp:i*temp)
END DO

j=0
k=0.0
DO WHILE(j<temp+2)
   j=j+1
   x_mod(1,j)      = f0( k/(temp+1) )
   x_mod(j,1)      = f1( k/(temp+1) )
   x_mod(temp+2,j) = f2( k/(temp+1) )
   x_mod(j,temp+2) = f3( k/(temp+1) )
   k = k + 1.0
END DO

ALLOCATE(g_mod((temp+2)*(temp+2)))

g_mod = 0

i=0
DO WHILE(i<temp+1)
   i=i+1
   j=0
   DO WHILE(j<temp+1)
      j=j+1
      pt1  = (j-1) * (temp+2) + i
      pt2  = pt1+1
      pt3  = pt2 + (temp+2)
      pt4  = pt3-1

      p=x_mod(i,j)
      q=x_mod(i+1,j)
      r=x_mod(i+1,j+1)
      s=x_mod(i,j+1)

      c1=SQRT(1+ ((s-p)/h)**2    + ((s-r)/h)**2)
      c2=SQRT(1+ ((p-q)/h)**2    + ((r-q)/h)**2)

      g_mod(pt1) = g_mod(pt1) + ( (p-s)/c1 + (p-q)/c2 )/2
      g_mod(pt2) = g_mod(pt2) + (2*q-p-r)/2/c2
      g_mod(pt3) = g_mod(pt3) + ( (r-s)/2/c1 + (r-q)/2/c2 )
      g_mod(pt4) = g_mod(pt4) + (2*s-p-r)/2/c1
```

```
    END DO
  END DO

  i=0
  DO WHILE(i<temp)
    i=i+1
    g(1+(i-1)*temp:i*temp) = g_mod(2+(i)*(temp+2):(i+1)*(temp+2)-1)
  END DO

  DEALLOCATE(x_mod)
  DEALLOCATE(g_mod)

END SUBROUTINE grad
```

```
SUBROUTINE hessian(x, Hval, Hrow, Hcol, Hnz)
  USE TYPES,ONLY : wp

  REAL ( KIND = wp ), INTENT(IN), DIMENSION( : ) :: x
  REAL ( KIND = wp ), POINTER, DIMENSION( : )    :: Hval
  INTEGER, POINTER, DIMENSION( : )               :: Hrow
  INTEGER, POINTER, DIMENSION( : )               :: Hcol
  INTEGER, INTENT(OUT)                           :: Hnz

  ! Local variables
  REAL( KIND = wp ) :: a,h,p,q,r,s,c1,c2, k
  REAL( KIND = wp ) :: xmt, zmt, xmy, zmy, dtmxmz, dymxmz
  INTEGER :: i,j,temp, ind, tmp_Hnz, pt1, pt2, pt3, pt4
  REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( :, : ) :: x_mod
  REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( : )    :: g1,g2,DD0
  REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( : )    :: DD1,DD2,DD3
  REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( : )    :: tmp_Hval
  INTEGER, ALLOCATABLE, DIMENSION( : )               :: tmp_Hrow
  INTEGER, ALLOCATABLE, DIMENSION( : )               :: tmp_Hcol

  a = SIZE(x)
  a = SQRT(a)
  temp = NINT(a)
  h=1/(a+1)

  ALLOCATE(x_mod(temp+2,temp+2))
  x_mod = 0
  i = 0
  DO WHILE(i<temp)
    i = i+1
    x_mod(2:temp+1,i+1) = x(1+(i-1)*temp:i*temp)
  END DO
```

```
j=0
k=0.0
DO WHILE(j<temp+2)
   j=j+1
   x_mod(1,j)      = f0( k/(temp+1.0) )
   x_mod(j,1)      = f1( k/(temp+1.0) )
   x_mod(temp+2,j) = f2( k/(temp+1.0) )
   x_mod(j,temp+2) = f3( k/(temp+1.0) )
   k = k + 1.0
END DO

ALLOCATE(g1(4))
ALLOCATE(g2(4))
ALLOCATE(DD0((temp+2)*(temp+2)))
ALLOCATE(DD1(size(DD0)-1))
ALLOCATE(DD2(size(DD0)-(temp+2)))
ALLOCATE(DD3(size(DD0)-(temp+3)))

DD0 = 0
DD1 = 0
DD2 = 0
DD3 = 0

i=0
DO WHILE(i<temp+1)
   i=i+1
   j=0
   DO WHILE(j<temp+1)
      j=j+1

      pt1  = (j-1) * (temp+2) + i
      pt2  = pt1+1
      pt3  = pt2 + (temp+2)
      pt4  = pt3-1

      p=x_mod(i,j)
      q=x_mod(i+1,j)
      r=x_mod(i+1,j+1)
      s=x_mod(i,j+1)

      c1=SQRT(1+ ((s-p)/h)**2    + ((s-r)/h)**2)
      c2=SQRT(1+ ((p-q)/h)**2    + ((r-q)/h)**2)

      g1(1) = 1/((h)**2)/c1*(p-s)
```

```
        g1(2) = 0
        g1(3) = 1/((h)**2)/c1*(r-s)
        g1(4) = 1/((h)**2)/c1*(2*s-p-r)

        g2(1) = 1/((h)**2)/c2*(p-q)
        g2(2) = 1/((h)**2)/c2*(2*q-p-r)
        g2(3) = 1/((h)**2)/c2*(r-q)
        g2(4) = 0

        xmt=p-s
        zmt=r-s
        dtmxmz=2*s-p-r
        xmy=p-q
        zmy=r-q
        dymxmz=2*q-p-r

        DD0(pt1) = DD0(pt1) + c1/(c1**2)/2-g1(1)*xmt/(c1**2)/2+     &
             c2/(c2**2)/2-g2(1)*xmy/(c2**2)/2
        DD0(pt2) = DD0(pt2) + 2*c2/(c2**2)/2-g2(2)*dymxmz/(c2**2)/2
        DD0(pt3) = DD0(pt3) + c1/(c1**2)/2-g1(3)*zmt/(c1**2)/2+     &
             c2/(c2**2)/2-g2(3)*zmy/(c2**2)/2
        DD0(pt4) = DD0(pt4) + 2*c1/(c1**2)/2-g1(4)*dtmxmz/(c1**2)/2
        DD1(pt1) = DD1(pt1) - g1(2)*xmt/(c1**2)/2-c2/(c2**2)/2      &
             -g2(2)*xmy/(c2**2)/2
        DD2(pt1) = DD2(pt1) - c1/(c1**2)/2-g1(4)*xmt/(c1**2)/2      &
             -g2(4)*xmy/(c2**2)/2
        DD3(pt1) = DD3(pt1) - g1(3)*xmt/(c1**2)/2-g2(3)*xmy/(c2**2)/2
        DD2(pt2) = DD2(pt2) - c2/(c2**2)/2-g2(3)*dymxmz/(c2**2)/2
        DD1(pt4) = DD1(pt4) - c1/(c1**2)/2-g1(4)*zmt/(c1**2)/2      &
             -g2(4)*zmy/(c2**2)/2
    END DO
END DO

tmp_Hnz = 2*size(DD3) + 2*size(DD2) + 2*size(DD1) + size(DD0)

IF(ASSOCIATED(Hval)) DEALLOCATE(Hval)
IF(ASSOCIATED(Hrow)) DEALLOCATE(Hrow)
IF(ASSOCIATED(Hcol)) DEALLOCATE(Hcol)

ALLOCATE(tmp_Hval(tmp_Hnz))
ALLOCATE(tmp_Hrow(tmp_Hnz))
ALLOCATE(tmp_Hcol(tmp_Hnz))

Hnz = 0
```

```
ind = 1
i=0
DO WHILE(i<size(DD3))
   i=i+1
   tmp_Hval(ind) = DD3(i)
   tmp_Hrow(ind) = i
   tmp_Hcol(ind) = (temp+3)+i
   ind = ind +1
END DO

i=0
DO WHILE(i<size(DD3))
   i=i+1
   tmp_Hval(ind) = DD3(i)
   tmp_Hrow(ind) = (temp+3)+i
   tmp_Hcol(ind) = i
   ind = ind +1
END DO

i=0
DO WHILE(i<size(DD0))
   i=i+1
   tmp_Hval(ind) = DD0(i)
   tmp_Hrow(ind) = i
   tmp_Hcol(ind) = i
   ind = ind +1
END DO

i=0
DO WHILE(i<size(DD1))
   i=i+1
   tmp_Hval(ind) = DD1(i)
   tmp_Hrow(ind) = i
   tmp_Hcol(ind) = i+1
   ind = ind +1
END DO

i=0
DO WHILE(i<size(DD1))
   i=i+1
   tmp_Hval(ind) = DD1(i)
   tmp_Hrow(ind) = i+1
   tmp_Hcol(ind) = i
   ind = ind +1
END DO
```

```
i=0
DO WHILE(i<size(DD2))
   i=i+1
   tmp_Hval(ind) = DD2(i)
   tmp_Hrow(ind) = i
   tmp_Hcol(ind) = (temp+2)+i
   ind = ind +1
END DO

i=0
DO WHILE(i<size(DD2))
   i=i+1
   tmp_Hval(ind) = DD2(i)
   tmp_Hrow(ind) = (temp+2)+i
   tmp_Hcol(ind) = i
   ind = ind +1
END DO

i=0
Hnz = 0
DO WHILE(i<size(tmp_Hval))
   i=i+1
   IF((1<mod(tmp_Hrow(i),temp+2)).AND.&
        ((temp+2)>mod(tmp_Hrow(i),temp+2)).AND. &
        (0<((tmp_Hrow(i)-mod(tmp_Hrow(i),temp+2))/(temp+2))) &
        .AND. ((temp+1)>((tmp_Hrow(i)-  &
        mod(tmp_Hrow(i),temp+2))/(temp+2)))  ) THEN
      IF((1<mod(tmp_Hcol(i),temp+2)).AND.&
           ((temp+2)>mod(tmp_Hcol(i),temp+2)).AND. &
           (0<((tmp_Hcol(i)-mod(tmp_Hcol(i),temp+2))/(temp+2)))&
           .AND. ((temp+1)>((tmp_Hcol(i)-&
           mod(tmp_Hcol(i),temp+2))/(temp+2)))  ) THEN
         Hnz = Hnz + 1
      END IF
   END IF
END DO

ALLOCATE(Hval(Hnz))
ALLOCATE(Hrow(Hnz))
ALLOCATE(Hcol(Hnz))

i=0
ind = 1
DO WHILE(i<size(tmp_Hval))
```

```
     i=i+1
     IF((1<mod(tmp_Hrow(i),temp+2)).AND.&
          ((temp+2)>mod(tmp_Hrow(i),temp+2)).AND. &
          (0<((tmp_Hrow(i)-mod(tmp_Hrow(i),temp+2))/(temp+2)))&
          .AND. ((temp+1)>((tmp_Hrow(i)- &
          mod(tmp_Hrow(i),temp+2))/(temp+2)))  ) THEN
        IF((1<mod(tmp_Hcol(i),temp+2)).AND.&
             ((temp+2)>mod(tmp_Hcol(i),temp+2)).AND. &
             (0<((tmp_Hcol(i)-mod(tmp_Hcol(i),temp+2))/(temp+2)))&
             .AND. ((temp+1)>((tmp_Hcol(i)-&
             mod(tmp_Hcol(i),temp+2))/(temp+2)))  ) THEN
           Hval(ind) = tmp_Hval(i)
           Hrow(ind) = tmp_Hrow(i)-(temp+2)-1-&
                2*(((tmp_Hrow(i)-&
                mod(tmp_Hrow(i),temp+2))/(temp+2))-1)
           Hcol(ind) = tmp_Hcol(i)-(temp+2)-1-&
                2*(((tmp_Hcol(i)-&
                mod(tmp_Hcol(i),temp+2))/(temp+2))-1)
           ind=ind+1
        END IF
     END IF
  END DO

  DEALLOCATE(tmp_Hval)
  DEALLOCATE(tmp_Hrow)
  DEALLOCATE(tmp_Hcol)

  DEALLOCATE(x_mod)
  DEALLOCATE(g1)
  DEALLOCATE(g2)
  DEALLOCATE(DD0)
  DEALLOCATE(DD1)
  DEALLOCATE(DD2)
  DEALLOCATE(DD3)

END SUBROUTINE hessian
```

```
SUBROUTINE compute_lower_bound(lb)
  USE TYPES,ONLY : wp

  REAL ( KIND = wp ), INTENT(INOUT), DIMENSION( : ) :: lb

  ! Local variables
  INTEGER :: i, j, k, temp, temp2
  REAL ( KIND = wp ) :: a
  REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( : ) :: tmp
```

```
  lb = 0.0
  temp = size(lb)
  a = SQRT(DBLE(temp))
  temp = NINT(a)
  ALLOCATE(tmp(temp))
  tmp = 0.0
  temp2 = (temp-MODULO(temp,9))/9
  IF(temp2>1) THEN
     i = 4*temp2 - 1
     DO WHILE(i<5*temp2)
        i=i+1
        tmp(i) = SQRT(2.0)
     END DO
  END IF
  k = 1
  i = 0
  DO WHILE(i<size(tmp))
     i=i+1
     j = 0
     DO WHILE(j<size(tmp))
        j=j+1
        lb(k) = tmp(i)*tmp(j)
        k = k + 1
     END DO
  END DO
  DEALLOCATE(tmp)

END SUBROUTINE compute_lower_bound
```

This example is the one implemented in the `rmtrs` program. The RMTR algorithm is launched with the following control and problem specification files:

```
BEGIN RUNRMTR.SPC
error-printout-device                                   6
printout-device                                         6
print-level                                       SUMMARY
criticality-threshold                                 0.001
truncated-conjugate-gradient-accuracy                   0.1
maximum-number-of-iterations                         1000
maximum-number-of-tcg-iterations                        5
initialization-technique                               FM
display-equivalent-evaluations                          T
cycling-style                                     Vcycles
minimum-rho-for-successful-iteration                  0.01
minimum-rho-for-very-successful-iteration               0.9
```

```
radius-reduction-factor                                    0.25
radius-increase-factor                                     2.0
initial-radius                                             1.0
coarse-model-choice-parameter                              0.25
linesearch                                                 2
model-backtracking                                         T
quadratic-model                                       GALERKIN
forced-hessian-estimation-frequency                        0
forced-hessian-evaluation-factor                           0.5
euclidean-gradient-accuracy-for-hessian-evaluation         0.15
infinite-gradient-accuracy-for-hessian-evaluation      10000.0
number-of-smoothing-cycles                                 7
smooth-frequency                                  ALWAYS_SMOOTH
start-printing-at-iteration                                0
stop-printing-at-iteration                                -1
maximum-radius                                             1.0D20
maximum-radius-increase-factor                             3.0
save-solution                                              T
display-options                                            F
checkpointing-frequency                                    0
checkpointing-file                                       RMTR.sav
checkpointing-device                                      55
restart-from-checkpoint                                    F
maximum-solving-time                                      -1
END RUNRMTR.SPC

BEGIN PROBLEM.SPC
problem-dimension 2
level-min 1
level-max 7
interpolation-type 2
approximate-hessian EXACT_HESSIAN
number-of-coarsest-level-discretization-points 9
upper-bound F
lower-bound T
predefined-sparsity-pattern 1
starting-point-file RMTR_startingpoint.dat
solution-file RMTR_solution.dat
END PROBLEM.SPC
```

Here is the output for the problem.

```
 Iterations summary:
 -------------------
```

```
Level               :      1     2     3     4     5     6     7
-------------------------------------------------------------------

Taylor              :    121     0     0     0     0     0     0
Taylor iterations   :    613     0     0     0     0     0     0
Smoothing           :      0   502   580   472   276   193   161
Cycles              :      0   544   623   472   276   193   161
Model-Backtracking  :      0     0     2     0    17    21    19
f evaluations       :      7     7    13   229   227   164   250
g evaluations       :      7     7    11   229   210   143   231
H evaluations       :      6     4     8     3    18    22    18
H updates           :      0     0     0     0     0     0     0
H reductions        :      0     6    25    23    23    14     6
Prolongations       :    115   290   291   179    82    69     0
Restrictions        :      0  1075  1765  1751  1131   657   779
Projections         :      0     0     0     0     0     0     0
Rejected projections:      0     0     0     0     0     0     0


 Equivalent evaluations number
--------------------------------
f evaluations      :   308.8250
g evaluations      :   283.5046
H evaluations      :    24.7085
smoothing cycles   :   236.8398
Taylor iterations  :     0.1497
H updates          :     0.0000
H eval+upd         :    24.7085


Total CPU time    :     24.922 second(s)
Solving time      :     24.722 second(s)

          ********************** Bye **********************
```

Bibliography

B. M. Averick and J. J. Moré. The Minpack-2 test problem collection. Technical Report ANL/MCS-TM-157, Mathematics and Computer Science, Argonne National Laboratory, Argonne, Illinois, USA, 1991.

R. E. Bank, P. E. Gill, and R. F. Marcia. Interior point methods for a class of elliptic variational inequalities. *in* T. Biegler, O. Ghattas, M. Heinkenschloss and B. Van Bloemen Waanders, eds, 'High performance algorithms and software for nonlinear optimization', pp. 218–235, Heidelberg, Berlin, New York, 2003. Springer Verlag.

F. Bastin, C. Cirillo, and Ph. L. Toint. Application of an adaptive Monte-Carlo algorithm to mixed logit estimation. *Transportation Research B*, **(to appear)**, 2006*a*.

F. Bastin, C. Cirillo, and Ph. L. Toint. Convergence theory for nonconvex stochastic programming with an application to mixed logit. *Mathematical Programming, Series A*, **108**(2-3), 207–234, 2006*b*.

F. Bastin, V. Malmedy, M. Mouffe, Ph. L. Toint, and D. Tomanos. A retrospective trust-region method for unconstrained optimization. *Mathematical Programming*, **(to appear)**, 2009.

S. J. Benson, L. C. McInnes, J. Moré, and J. Sarich. Scalable algorithms in optimization: Computational experiments. Preprint ANL/MCS-P1175-0604, Mathematics and Computer Science, Argonne National Laboratory, Argonne, Illinois, USA, 2004. To appear in the Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization (MA&O) Conference, August 30 - September 1, 2004.

J. T. Betts and S. O. Erb. Optimal low thurst trajectory to the moon. *SIAM Journal on Applied Dynamical Systems*, **2**(2), 144–170, 2003.

A. Borzi and K. Kunisch. A globalisation strategy for the multigrid solution of elliptic optimal control problems. *Optimization Methods and Software*, **21**(3), 445–459, 2006.

J. H. Bramble. *Multigrid Methods*. Longman Scientific and Technical, New York, 1993.

A. Brandt. Multi-level adaptative technique (mlat) for fast numerical solution to boundary value problems. *Proc. 3rd Int. Conf. on Numerical Methods in Fluid Mechanics*, **1**, 82–89, 1973.

A. Brandt. Multi-level adaptative solutions to boundary value problems. *Mathematics of Computation*, **31**(138), 333–390, 1977.

W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, Philadelphia, USA, 2nd edn, 2000.

M. M. Bronstein, A. M. Bronstein, R. Kimmel, and I. Yavneh. A multigrid approach for multi-dimensional scaling. Talk at the 12th Copper Mountain Conference on Multigrid Methods, 2005.

C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and its Applications*, **6**, 76–90, 1970.

C. Cartis, N. I. M. Gould, and Ph. L. Toint. Trust-region and other regularisations of linear least-squares problems. *BIT*, **49**(1), 21–53, 2009.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. Number 01 *in* 'MPS-SIAM Series on Optimization'. SIAM, Philadelphia, USA, 2000.

A. R. Conn, N. I. M. Gould, A. Sartenaer, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization using inexact projections on convex constraints. *SIAM Journal on Optimization*, **3**(1), 164–221, 1993.

A. R. Conn, L. N. Vicente, and C. Visweswariah. Two-step algorithms for nonlinear optimization with structured applications. *SIAM Journal on Optimization*, **9**(4), 924–947, 1999.

W. C. Davidon. Variance algorithms for minimization. *Computer Journal*, **10**, 406–410, 1968.

R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact-Newton methods. *SIAM Journal on Numerical Analysis*, **19**(2), 400–408, 1982.

J. E. Dennis. A brief introduction to quasi-Newton methods. *in* G. H. Golub and J. Oliger, eds, 'Numerical Analysis', number 22 *in* 'Proceedings of Symposia in Applied Mathematics', pp. 19–52, Providence, RI, USA, 1978. American Mathematical Society.

J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1983. Reprinted as *Classics in Applied Mathematics 16*, SIAM, Philadelphia, USA, 1996.

M. Do Carmo. *Differential geometry of curves and surfaces*. Prentice Hall, 1976.

E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, **91**(2), 201–213, 2002.

H. Sue Dollar, Nicholas I. M. Gould, and Daniel P. Robinson. On solving trust-region and other regularised subproblems in optimization. Technical Report NA-09/01, Oxford University Computing Laboratory, February 2009.

M. Domorádová and Z. Dostál. Projector preconditioning for bound-constrained quadratic optimization. *Linear Algebra and its Applications*, **to appear**, 2007.

M. Emilianenko. A nonlinear energy-based multilevel quantization scheme. Talk at the 12th Copper Mountain Conference on Multigrid Methods, 2005.

R.P. Fedorenko. The speed of convergence of one iterative process. *USSR Comput. Math. and Math. Phys.*, **4**(3), 227–235, 1964.

A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. J. Wiley and Sons, Chichester, England, 1968. Reprinted as *Classics in Applied Mathematics 4*, SIAM, 1990.

M. Fisher. Minimization algorithms for variational data assimilation. *in* 'Recent Developments in Numerical Methods for Atmospheric Modelling', pp. 364–385. ECMWF, 1998.

R. Fletcher. An efficient, globally convergent, algorithm for unconstrained and linearly constrained optimization problems. Technical Report TP 431, AERE Harwell Laboratory, Harwell, Oxfordshire, England, 1970.

R. Fletcher. *Practical Methods of Optimization*. J. Wiley and Sons, Chichester, England, second edn, 1987.

E. Fontdecaba i Baig. *High performance algorithms for progressive addition lens design*. PhD thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, 2000.

D. M. Gay. Computing optimal locally constrained steps. *SIAM Journal on Scientific and Statistical Computing*, **2**, 186–197, 1981.

E. Gelman and J. Mandel. On multilevel iterative methods for optimization problems. *Mathematical Programming*, **48**(1), 1–17, 1990.

E. M. Gertz. *Combination Trust-Region Line-Search Methods for Unconstrained Optimization*. PhD thesis, Department of Mathematics, University of California, San Diego, California, USA, 1999.

P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, London, 1981.

D. Goldfarb. A family of variable metric methods derived by variational means. *Mathematics of Computation*, **24**, 23–26, 1970.

G. H. Golub and C. F. Van Loan. *Matrix computations*. North Oxford Academic, Oxford, UK, 1983.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, second edn, 1989.

N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr, a constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, **29**(4), 373–394, 2003*a*.

N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software*, **29**(4), 353–372, 2003*b*.

N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint. Sensitivity of trust-region algorithms on their parameters. *4OR, Quarterly Journal of the Italian, French and Belgian OR Societies*, **3**(3), 227–241, 2005.

S. Gratton, M. Mouffe, A. Sartenaer, Ph. L. Toint, and D. Tomanos. Numerical experience with a recursive trust-region method for multilevel nonlinear optimization. *Optimization Methods and Software*, **(to appear)**, 2009.

S. Gratton, M. Mouffe, Ph. L. Toint, and M. Weber-Mendonca. A recursive trust-region method in infinity norm for bound-constrained nonlinear optimization. *IMA Journal of Numerical Analysis*, **28**(4), 827–861, 2008*a*.

S. Gratton, A. Sartenaer, and Ph. L. Toint. Numerical experience with a recursive trust-region method for multilevel nonlinear optimization. Technical Report 06/01, Department of Mathematics, University of Namur, Namur, Belgium, 2006.

S. Gratton, A. Sartenaer, and Ph. L. Toint. Recursive trust-region methods for multiscale nonlinear optimization. *SIAM Journal on Optimization*, **19**(1), 414–444, 2008*b*.

A. Griewank. The modification of newton's method for unconstrained optimization by bounding cubic terms. Technical Report DAMTP/NA12, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 1981.

A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. *in* M. J. D. Powell, ed., 'Nonlinear Optimization 1981', pp. 301–312, London, 1982. Academic Press.

W. A. Gruver and E. Sachs. *Algorithmic Methods in Optimal Control*. Pitman, Boston, USA, 1980.

W. Hackbusch. Ein iteratives verfharen zur schnellen auflosung elliptischer randwertprobleme. Report 76-12, Universitat Koln, 1976.

W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer Series in Applied Mathematical Sciences. Springer Verlag, Heidelberg, Berlin, New York, 1994.

W. Hackbusch. *Multi-grid Methods and Applications*. Number 4 *in* 'Springer Series in Computational Mathematics'. Springer Verlag, Heidelberg, Berlin, New York, 1995.

M. D. Hebden. An algorithm for minimization using exact second derivatives. Technical Report T.P. 515, AERE Harwell Laboratory, Harwell, Oxfordshire, England, 1973.

P. W. Hemker and G. M. Johnson. Multigrid approach to Euler equations. *in* S. F. McCormick, ed., 'Multigrid methods', Vol. 3 of *Frontiers in Applied Mathematics*, pp. 57–72, Philadelphia, USA, 1987. SIAM.

M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of the National Bureau of Standards*, **49**, 409–436, 1952.

K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly Journal on Applied Mathematics*, **2**, 164–168, 1944.

M. Lewis and S. G. Nash. Model problems for the multigrid optimization of systems governed by differential equations. *SIAM Journal on Scientific Computing*, **26**(6), 1811–1837, 2005.

D. G. Luenberger. *Optimization by Vector Space Methods*. J. Wiley and Sons, Chichester, England, 1969.

J. Mandel and S. McCormick. A multilevel variational method for $Au = \lambda Bu$ on composite grids. *J. Comput. Phys.*, **80**(2), 442–452, 1989.

O. L. Mangasarian. *Nonlinear Programming*. McGraw-Hill, New York, USA, 1979.

D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, **11**, 431–441, 1963.

J. J. Moré and D. C. Sorensen. On the use of directions of negative curvature in a modified Newton method. *Mathematical Programming*, **16**(1), 1–20, 1979.

J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, **4**(3), 553–572, 1983.

J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, **7**(1), 17–41, 1981.

D. D. Morrison. Methods for nonlinear least squares problems and convergence proofs. *in* J. Lorell and F. Yagi, eds, 'Proceedings of the Seminar on Tracking Programs and Orbit Determination', pp. 1–9, Pasadena, USA, 1960. Jet Propulsion Laboratory.

M. Mouffe. *Multilevel optimization in infinity norm and associated stopping criteria*. PhD thesis, CERFACS, Toulouse, France, 2009.

S. G. Nash. A multigrid approach to discretized optimization problems. *Optimization Methods and Software*, **14**, 99–116, 2000.

Y. Nesterov and B. T. Polyak. Cubic regularization of Newton method and its global performance. *Mathematical Programming*, **108**(1), 177–205, 2006.

J. Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, **35**, 773–782, 1980.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Series in Operations Research. Springer Verlag, Heidelberg, Berlin, New York, 1999.

J. Nocedal and Y. Yuan. Combining trust region and line search techniques. *in* Y. Yuan, ed., 'Advances in Nonlinear Programming', pp. 153–176, Dordrecht, The Netherlands, 1998. Kluwer Academic Publishers.

B. O'Neill. *Elementary differential geometry (second edition)*. Academic Press, 1997.

J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, London, 1970.

A. Perry. A modified conjugate gradient algorithm. Technical Report 229, Center for Mathematical Studies in Economics and Management Science, Northwestern University, 1976.

M. J. D. Powell. A new algorithm for unconstrained optimization. *in* J. B. Rosen, O. L. Mangasarian and K. Ritter, eds, 'Nonlinear Programming', pp. 31–65, London, 1970. Academic Press.

M. J. D. Powell and Ph. L. Toint. On the estimation of sparse Hessian matrices. *SIAM Journal on Numerical Analysis*, **16**(6), 1060–1074, 1979.

S. M. Robinson. Analysis of sample-path optimization. *Mathematics of Operations Research*, **21**(3), 513–528, 1996.

R. T. Rockafellar. *Convex Analyis*. Princeton University Press, Princeton, USA, 1970.

A. Sartenaer. Automatic determination of an initial trust region in nonlinear programming. *SIAM Journal on Scientific Computing*, **18**(6), 1788–1803, 1997.

D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, **24**, 647–657, 1970.

D. F. Shanno. Conjugate gradient methods with inexact searches. *Mathematics of Operations Research*, **3**, 244–256, 1978.

A. Shapiro. Monte Carlo sampling methods. *Handbooks in Operations Research and Management*, **10**, 353–425, 2003.

D. C. Sorensen. Newton's method with a model trust-region modification. *SIAM Journal on Numerical Analysis*, **19**(2), 409–426, 1982.

T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, **20**(3), 626–637, 1983.

W. A. Sutherland. *Introduction to Metric and Topological Spaces*. Oxford University Press, Oxford, England, 1975.

Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. *in* I. S. Duff, ed., 'Sparse Matrices and Their Uses', pp. 57–88, London, 1981. Academic Press.

Ph. L. Toint. VE08AD, a routine for partially separable optimization with bounded variables. *Harwell Subroutine Library*, **2**, 1983.

Ph. L. Toint. VE10AD, a routine for large scale nonlinear least squares. *Harwell Subroutine Library*, **2**, 1987.

Ph. L. Toint and D. Tomanos. Optimisation numérique appliquée, le design des lentilles progressives. *Tangente*, **126**, 48–50, 2009.

Ph. L. Toint, D. Tomanos, and M. Weber-Mendonca. A multilevel algorithm for solving the trust-region subproblem. *Optimization Methods and Software*, **24**(2), 299–311, 2009.

U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Elsevier, Amsterdam, The Netherlands, 2001.

S. A. Vavasis and R. Zippel. Proving polynomial-time for sphere-constrained quadratic programming. Technical Report TR 90-1182, Department of Computer Science, Cornell University, Ithaca, New York, USA, 1990.

M. Weiser, P. Deuflhard, and B. Erdmann. Affine conjugate adaptive newton methods for nonlinear elastomechanics. *Optimization Methods and Software*, **22**(3), 413–431, 2007.

Zaiwen Wen and Donald Goldfarb. A linesearch multigrid method ofr large-scale convex optimization. *SIAM Journal on Optimization*, **to appear**, 2008.

P. Wesseling. *An introduction to Multigrid Methods*. J. Wiley and Sons, Chichester, England, 1992. Corrected Reprint, Edwards, Philadelphia, 2004.

Y. Yuan. On the truncated conjugate-gradient method. *Mathematical Programming, Series A*, **87**(3), 561–573, 2000.